ГОСЫ ГЛАВА 1

# Дифференциальное исчисление

## 1 Непрерывность функции в точке по Коши и по Гейне. Доказательство эквивалентности определений.

**Непрерывность функции в точке по Коши:**

Функция *f*(*x*) называется непрерывной в точке *x*0, если она определена на некоторой окрестности *U*(*x*0) этой точки, и если, для любого сколь угодно малого положительного числа *ε* > 0, существует такое число *δε* > 0, зависящее от *ε*, что для всех *x*, принадлежащих *δε* - окрестности точки *x*0: |*x - x*0| < *δε* , значения функции принадлежат *ε* - окрестности точки *f*(*x*0): |*f*(*x*)-*f*(*x*0)|<*ε.*

**Запись с помощью логических символов:**

∀*ε* > 0, ∃*δε* > 0, ∀*x*  |*x - x*0| < *δε* : | *f*(*x*) - *f*(*x*0) |< *ε.*

**Непрерывность функции в точке по Гейне:**

Функция *f*(*x*) называется непрерывной в точке *x*0, если она определена на некоторой окрестности *U*(*x*0) этой точки, и если для любой последовательности {*xn*}, сходящейся к *x*0: *xn*  = *x*0 (n -> ∞), элементы которой принадлежат окрестности *U*(*x*0), последовательность {*f*(*xn*)} сходится к *f*(*x*0): *f*(*xn* ) =  *f*(*x*0) (n -> ∞) .

**Запись с помощью логических символов:**

∀{*xn*}: {*xn*} *U*(*x*0), *xn* = *x*0(n -> ∞) => *f*(*xn* )= *f*(*x*0) (n -> ∞).

**Теорема:** Определения непрерывности функции по Коши и по Гейне эквивалентны.

**Доказательство:**

(⇒) Покажем, что если выполняется определение непрерывности по Коши, то выполняется и определение по Гейне.

Пусть *f: E → R.*

Запишем определение непрерывности функции в точке по Коши в другом виде:

∀ε>0 ∃δ>0 ∀x∈ Bδ(a)∩E:f(x)∈ Bε (f(a)) (∗)

Пусть xn∈E xn →a, тогда ∃N ∀n > N: xn ∈ Bδ(a) ∩ E ⇒(∗) ∀n > N:

f(xn) ∈ Bε(f(a)).

Получим ∀ε>0 ∃N ∀n > N: f(xn) ∈ Bε(f(a)), т.е. f(xn)→f(a).

Определение по Гейне выполняется.

(⇐) Покажем, что если выполняется определение по Гейне, то выполняется и определение по Коши.

Если точка *a ∈ E* не является предельной точкой, то оба определения выполняются.

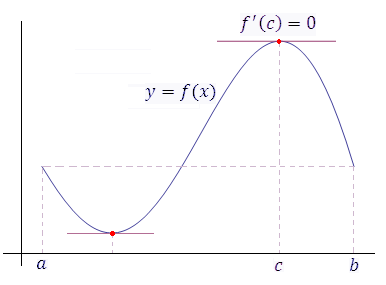
Если *a ∈ E* - предельная точка, то по определению предела функции по Гейне *lim f(x)=f(a) (Е*∈*x*→*a)*, а значит, *f* непрерывна в точке *a* в смысле определения Коши.

Ч.Т.Д.

## 2 Теоремы Ролля (формулировка), Лагранжа (с доказательством) и Коши (с доказательством) о средних значениях.

**Теорема Ролля.**

Пусть функция f(x) дифференцируема в открытом промежутке (a,b), на концах этого промежутка сохраняет непрерывность и принимает одинаковые значения: f(a) = f(b). Тогда существует точка c∈(a,b), в которой производная функции f(x) равна нулю: f’(c) = 0.



**Рис. 3**. Теорема Ролля устанавливает условия существования хотя бы одной точки *c*, в которой касательная к графику функции параллельна оси 0*x*. Таких точек может быть несколько.

**Теорема Лагранжа.**

Пусть функция f(x) дифференцируема в открытом промежутке (a,b) и сохраняет непрерывность на концах этого промежутка. Тогда существует такая точка c∈(a,b), что



**Доказательство**. Рассмотрим вспомогательную функцию



Эта функция непрерывна и дифференцируема в промежутке [a,b], а на его концах принимает одинаковые значения:

F(a) = f(b)=0.

Тогда F(x) удовлетворяет всем условиям теоремы Ролля и, следовательно, существует точка c∈(a,b), в которой производная функции F(X) равна нулю:



**Следствие 1**. В частном случае, когда f(b)=f(a), из теоремы Лагранжа вытекает, что существует точка c∈(a,b), в которой производная функции f(x) равна нулю: f’(c)=0. Это означает, что теорема Лагранжа является обобщением теоремы Ролля.

**Теорема Коши о средних значениях.**

**Теорема**. Пусть функции f(x) и g(x) непрерывны в замкнутом промежутке [a,b]; дифференцируемы в открытом промежутке (a,b); g’(x) ≠ 0 в открытом промежутке (a,b). Тогда существует такая точка c∈(a,b), что



**Доказательство**. Заметим, что g(b) - g(a) ≠ 0. В противном случае – согласно теореме Ролля – производная g’(x) обратилась бы в нуль в некоторой точке c∈(a,b).

Рассмотрим вспомогательную функцию



которая удовлетворяет всем условиям теоремы Ролля и, в частности, принимает одинаковые значения на концах промежутка [a,b]:

F(a)=F(b)=0.

Тогда существует точка c∈(a,b), в которой

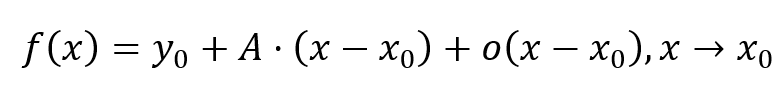


что и требовалось доказать.

**Следствие**. Теорема Лагранжа является частным случаем теоремы Коши при g(x)=x. В свою очередь теорема Ролля представляет собой частный случай теоремы Лагранжа. Таким образом, теорема Коши включает в себя в качестве частных случаев теорему Ролля и теорему Лагранжа.

## 3 Формулы Тейлора с остаточным членом в форме Лагранжа (с доказательством) и Пеано (формулировка).

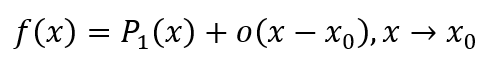
Если функция имеет в точке производную, то приращение этой функции представимо в виде , где , , и , т.е.



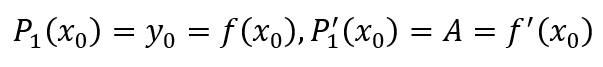
Иначе говоря, существует линейная функция

 (1)

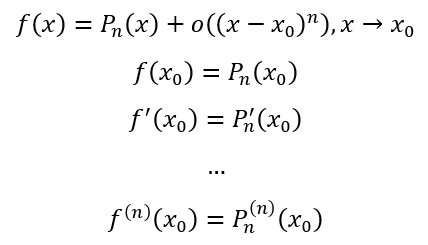
такая, что



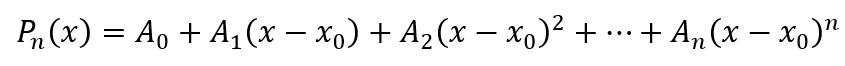
причём

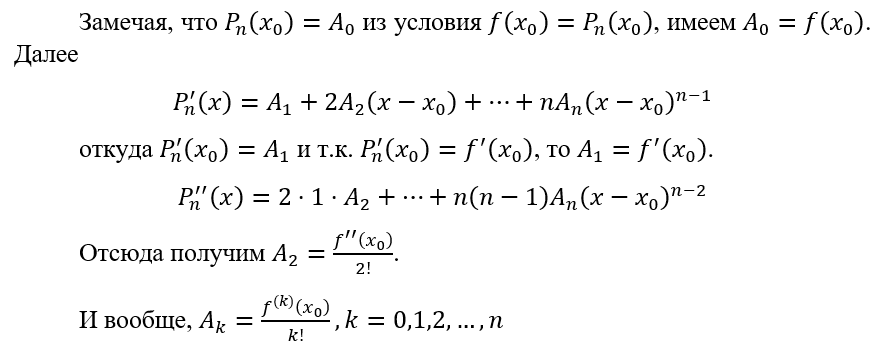


Поставим более общую задачу. Пусть функция имеет производных в точке . Требуется выяснить, существует ли многочлен степени не выше такой, что

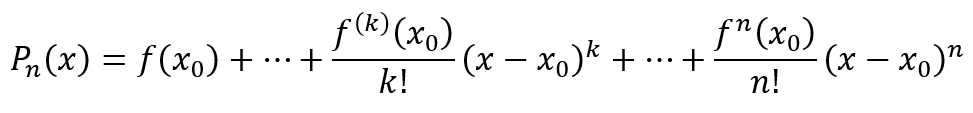


Будем искать этот многочлен по аналогии с формулой (1), в виде



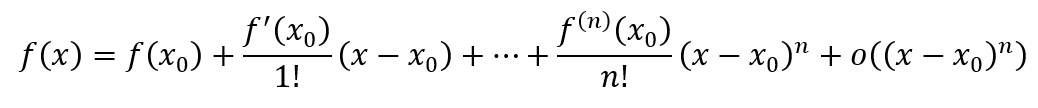


Тогда итоговый многочлен имеет вид

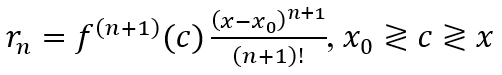


**Теорема**

Пусть функция , определённая на интервале имеет в точке производные до порядка включительно. Тогда при

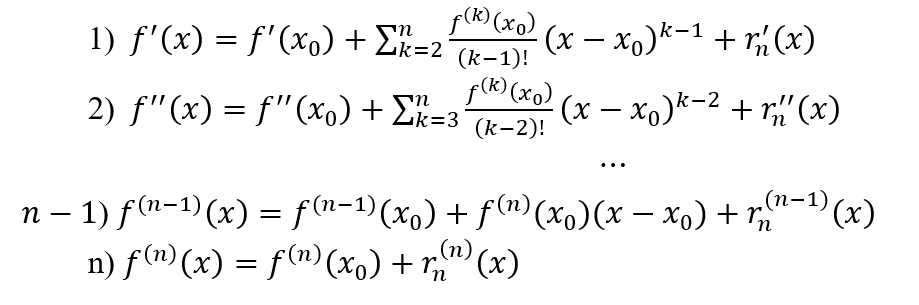


Данная формула называется формулой Тейлора n-го порядка с остаточным членом в форме Пеано, а функция – остаточным членом n-го порядка формулы Тейлора.

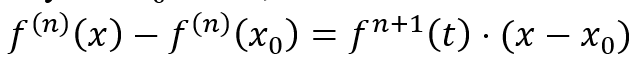
Форма Лагранжа – ,

Доказательство:

Продифференцируем по обе части формулы Тейлора раз:



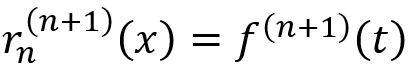
По теореме Лагранжа (есть в вопросе выше) существует точка , расположенная между и такая, что

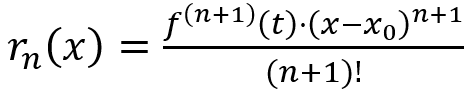


Отсюда

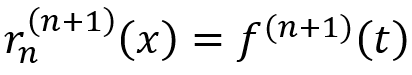


Продифференцируем ещё раз по и получим



Пусть остаточный член задан в виде 

Тогда он и все его производные равны нулю в точке и



# Интегральное исчисление

## 4 Правила вычисления определенных интегралов. Интегрирование по частям (формулировка и пример вычисления интеграла). Формула Ньютона-Лейбница (с доказательством).

Теорема.

Если есть некоторая первообразная от непрерывной функции , то справедлива формула: . Эта формула называется формулой Ньютона-Лейбница.

Доказательство.

Пусть есть некоторая первообразная . - тоже первообразная . Но любые две первообразные функции отличаются лишь на постоянную величину , следовательно . Постоянную величину легко определить, положив в равенстве получаем так как , то . ЧТД.

Для вычисления определенного интеграла необходимо вычислить неопределенный интеграл и воспользоваться формулой Ньютона-Лейбница.

Также, когда функция является определенной и непрерывной из отрезка [a, b], тогда имеющееся множество [a,b] считается областью значений функции , определенной на отрезке с имеющейся непрерывной производной, где и , отсюда получаем, что .

Если на отрезке [a, b] определены и непрерывны функции и , тогда из производные первого порядка являются интегрируемыми, таким образом из этого отрезка для интегрируемой функции равенство справедливо.

Пример:

Найдем неопределенный интеграл:

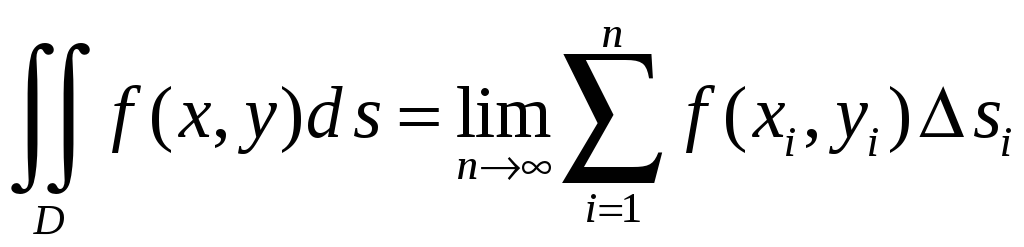
Следовательно:

Применим формулу Ньютона-Лейбница:

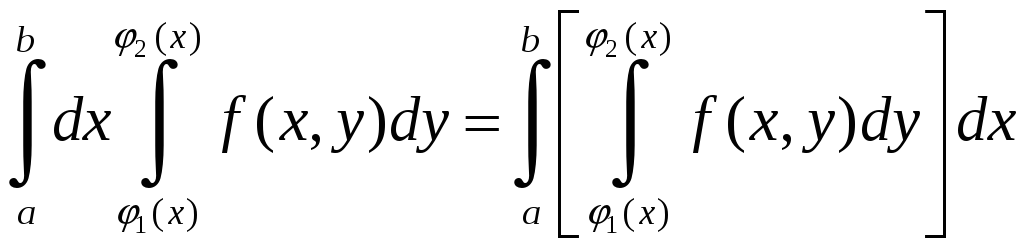
## 5 Определение двойного и повторного интегралов и связь между ними. Задача о нахождении массы тела. Определение тройного интеграла.

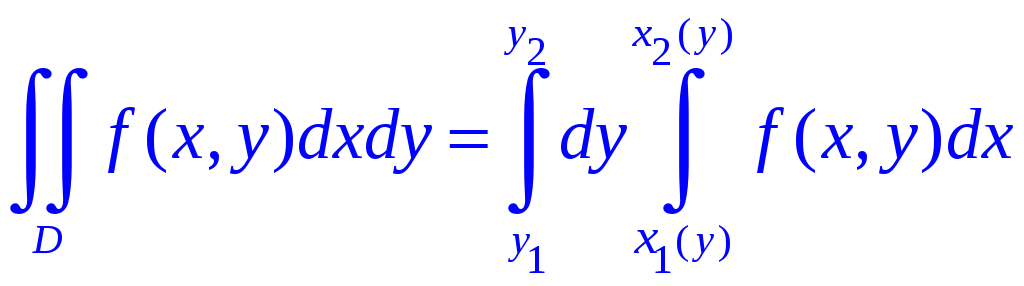
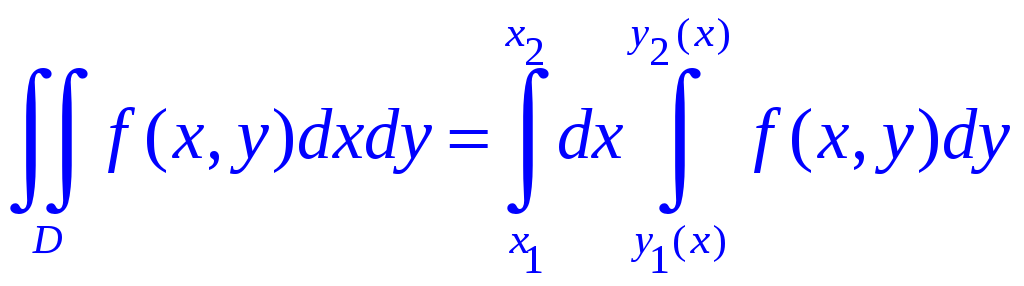
Двойной интеграл представляет собой обобщение понятия определенного интеграла на случай функции двух переменных. В этом случае вместо отрезка интегрирования будет присутствовать какая-то плоская фигура.

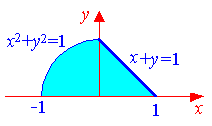
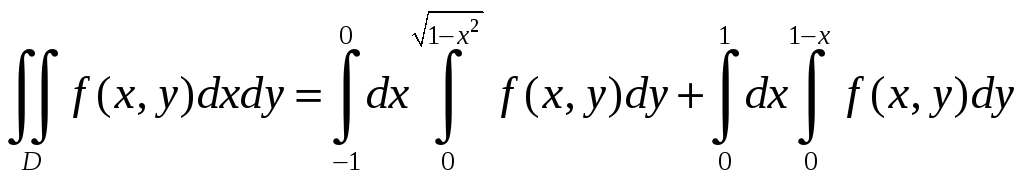


***Двойным интегралом*** *функции* *f*(*x,y*) *по области* *D* *называется предел, к которому стремится последовательность интегральных* *сумм Si = f(xi, yi)\*si, где si - площадь i-го участка, а xi,yi - некоторая его точка,* *при неограниченном увеличении числа разбиений* *n*.  
 

Повторными интегралами называются интегралы вида

В этом выражении сначала вычисляется внутренний интеграл, т.е. производится сначала интегрирование по переменной *y* (при этом переменная *x* считается постоянной величиной). В результате интегрирования по *y* получится некоторая функция по x. Её интегрируем в обычном интеграле и получаем численное значение.

Связь:  
Область называется *простой* в каком-либо направлении, если любая прямая, проведенная в этом направлении, пересекает границу области не более чем в двух точках. Любую непростую область можно разбить на сумму простых.  
***Теорема***. *Если область интегрирования D – простая в направлении оси Oy, то двойной интеграл можно записать в виде повторного:  
 , если она простая по Ох, то  
 , если простая везде, то любая форма записи подходит.*

(Доп инфа: Чтобы определить пределы интегрирования необходимо построить область D, выделить границы интегрирования по внешнему интегралу (это должны быть константы), а после определить границы внутреннего интеграла в виде функций от переменной внешнего. Вероятно, придётся прибегнуть к сумме интегралов для работы со всей зоной. Пример:  
Пусть D = , тогда)

Тройной интеграл является аналогом двойного интеграла и вводится для функции трех переменных. Пусть в пространственной области  определена и непрерывна функция трех переменных  . Разобьем область на  произвольных областей  v1, v2,…, vn с объемами  ,  ,…,  затем выберем в каждой области v*i* произвольную точку  и построим интегральную сумму вида

 .

Если интегральная сумма  имеет предел при стремлении к нулю наибольшего из диаметров областей  , то этот предел называется *тройным интегралом* и обозначается символом

 и 

Тройной интеграл обладает свойствами, аналогичными свойствам двойного интеграла (линейность, аддитивность, оценка интеграла, свойство среднего). Он вычисляется аналогично двойному - сведением к повторному.

Одно из приложений тройного интеграла - **вычисление массы пространственного тела** по формулам, которые выводятся аналогично соответствующим формулам для плоского тела с двойным интегралом. В данном случае нам необходимо определить функция f(x,y,z) - плотность вещества тела в каждой точке, после чего взять от неё тройной интеграл.

## 6 Понятие криволинейного интеграла первого рода, его геометрический и физический смысл. Сведение криволинейного интеграла первого рода к определенному интегралу.

**Определение криволинейного интеграла первого рода**

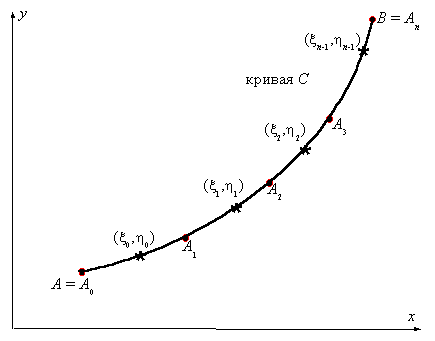
Пусть на плоскости *XOY* заданы

1. Некоторая кривая *С* с граничными точками *А* и *В*;

2. Функция двух переменных *f*(*x*, *y*), определенная, по крайней мере, на кривой *С*.

Проделаем следующую процедуру, которая является стандартной для построения определенного интеграла.

1. Разобьем кривую *С* на кусочки точками  и точку *А* будем считать точкой , а точку *В* – точкой .



Пусть  есть длина дуги кривой *С* между точками  и  и 

2. На каждом отрезке кривой *С* между точками  и  выберем произвольным образом «среднюю точку» с координатами  и составим интегральную сумму

.

3. Сделаем предельный переход при . Если существует  и он не зависит от способа разбиения кривой *С* на кусочки и от способа выбора средней точки, то он называется криволинейным интегралом первого рода от функции *f*(*x*, *y*) по кривой *С* и обозначается символом

.

Заметьте, что

.

**9.1.2 Вычисление криволинейного интеграла первого рода**

Пусть кривая *АВ* задана параметрически в виде

.

Тогда формула для вычисления криволинейного интеграла первого рода имеет вид

.

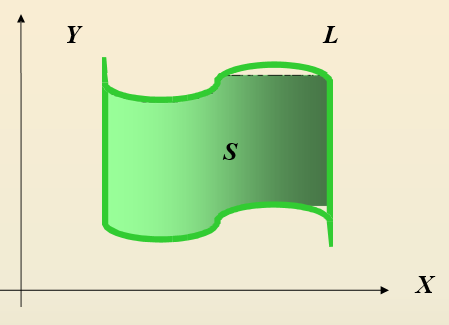
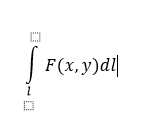
Если кривая задана явно в виде , то

.

Если кривая задана в полярных координатах , то



**Геометрический смысл**

**

*Площадь цилиндрической поверхности образованной кривой* ***L****, описываемой функцией* ***f(x; y)***

***Физ смысл***

*Пусть Γ   — гладкая кривая с параметризацией γ(t),t ∈ [a,b]  . Пусть, далее, f   — непрерывная вещественнозначная функция, определенная на Γ  . Интеграл*

| *∫        ∫b   fdl Оп=р. f(γ(t))|γ′(t)|dt Γ        a* |
| --- |

*не зависит от параметризации кривой и называется криволинейным интегралом 1 рода или интегралом по длине дуги. В частности, интеграл от f = 1   даст длину кривой Γ  . В координатном виде*

| *∘ ------------------- γ(t) = (x(t),y(t),z(t)),  |γ′(t)| =  x′2(t) +y′2(t)+ z′2(t).* |
| --- |

*Физический смысл интеграла 1 рода заключен в следующем. Если ρ   — линейная плотность тонкой тяжелой проволоки формы Γ  , то масса проволоки определяется интегралом*

| *∫ M =   ρdl.     Γ* |
| --- |

***Сведение к определенному***

***Пусть точки  являются концами линии , а сама она задана*** [***функцией одной переменной***](http://mathprofi.ru/mnozhestva.html) ***. Тогда криволинейный интеграл первого рода можно свести к обычному*** [***определённому интегралу***](http://mathprofi.ru/opredelennye_integraly_primery_reshenij.html) ***по следующей формуле:***

******

# Функциональные последовательности и ряды

## 7 Знакопеременные числовые ряды. Абсолютная сходимость. Свойства абсолютно сходящихся рядов. Условная сходимость. Признак Лейбница (с доказательством).

Ряд называется знакопеременным, если среди его слагаемых есть и положительные, и отрицательные.

Знакопеременный ряд называется знакочередующимся, если положительные и отрицательные его члены следуют друг за другом поочередно.

Знакочередующийся ряд сходится, если абсолютные величины его членов монотонно убывают с ростом порядкового номера, и общий член его стремится к нулю при n, стремящемся к бесконечности

Сходящийся ряд называется сходящимся абсолютно, если сходится ряд из модулей |иначе — сходящимся.

Свойства абсолютно сходящихся рядов:

Если ряд абсолютно сходится, то любой ряд, составленный из членов данного ряда, взятых, возможно, в другом порядке, тоже абсолютно сходится и имеет ту же сумму.

Если ряды sum(v1…vn) и sum(u1…un) абсолютно сходятся, то ряд, составленный из всевозможных попарных произведений*umvn*членов этих рядов, тоже абсолютно сходится, и его сумма равна произведению сумм исходных рядов.

Признак Лейбница — это признак сходимости знакочередующегося ряда.

Он заключается в следующем: если члены знакочередующегося ряда монотонно убывают и стремятся к нулю, то ряд сходится. При этом остаток ряда имеет знак своего первого члена и меньше его по абсолютной величине.

## 8 Равномерная сходимость функционального ряда. Признак Вейерштрасса с доказательством.

f1, f2, … равномерно сходится к f(x), если ∀ε>0 ∃N ∀n>N ∀x∈E:|fn(x)−f(x)|<ε Пишут, что fn⇉f

Пусть на E задан функциональный ряд ∑ (от n=1 до ∞) fn. Тогда он равномерно сходится к

f=∑fn, если ∀ε >0 ∃N ∀n>N ∀x∈E:|Sn(x)−f(x)|<ε

**Признак Вейерштрасса для проверки равномерной сходимости**

∑(от n=1 до ∞) fn, ∀n∈N, ∀x∈E:|fn(x)|≤an, ∑(от n=1 до ∞) an — сходится. Тогда

∑( отn=1 до ∞) fn равномерно сходится на E.

**Док-во:**

применим критерий Коши:

∣∑ (от k=n до m) fk(x)∣≤ ∑ (от k=n до m ) |fk(x)|≤ ∑ (от k=n до m) ak

∑(от k=n до m) ak <+∞⇒∀ε >0 ∃N ∀m≥n>N:∑(от k=n до m) ak<ε

Сопоставляя с предыдущим неравенством, которое верно ∀x,

∣∑ (от k=n до m) fk(x)∣<ε. Тогда по критерию Коши, ряд равномерно сходится.

## 9 Степенные ряды. Теоремы Абеля об абсолютной сходимости степенного ряда и о равномерной сходимости степенного ряда (с доказательствами). Круг сходимости степенного ряда.

картинки исправлю на текст

***Степенным рядом*** называется функциональный ряд вида

,

где числа, называемые коэффициентами ряда.

Областью сходимости степенного ряда всегда является некоторый интервал, который, в частности, может вырождаться в точку. Для того, чтобы убедиться в этом, докажем очень важную для всей теории степенных рядов теорему.

Теорема Абеля

Если степенной ряд сходится при некотором значении , не равном нулю, то он абсолютно сходится при всяком значении x, для которого ; если ряд расходится при некотором значении , то он расходится при всяком , для которого .

Доказательство. По условию теоремы числовой ряд

(12.1.38)

сходится, значит при  , а это значит, что , что все члены ряда по абсолютной величине меньше . Запишем ряд (12.1.38) в виде

 (12.1.39)

и рассмотрим ряд из абсолютных величин его членов

 (12.1.40)

Члены ряда (12.1.40) меньше соответствующих членов ряда

(12.1.41)

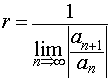
Ряд (12.1.41) при  представляет геометрическую прогрессию со знаменателем  и, следовательно, сходится. По признаку сравнения числовых рядов из сходимости ряда (12.1.41) следует сходимость ряда (12.1.40).

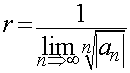
Из сходимости ряда (12.1.40) следует абсолютная сходимость ряда (12.1.39), а следовательно, и ряда (12.1.37). Значит, при  ряд (12.1.37) сходится абсолютно.

Докажем вторую часть теоремы. Пусть в некоторой точке  ряд (12.1.37) расходится. Тогда он будет расходиться и в любой точке , удовлетворяющей условию . Действительно, если бы в какой-либо точке , удовлетворяющей этому условию, ряд сходился, то в силу доказанного он должен был бы сходиться и в точке , так как , что противоречит условию. Следовательно, ряд расходится и в точке x. Таким образом, теорема полностью доказана.

**Радиус сходимости степенного ряда.** Так называют радиус круга сходимости [степенного ряда](http://school-collection.edu.ru/dlrstore-wrapper/Ryad/index.html)  на комплексной плоскости (или степенного ряда  на действительной числовой оси), т.е. такое число *r*, что ряд сходится при |*z*| < *r* (соответственно при |*x*| < *r*) и расходится при |*z*| > *r* (соответственно при |*x*| > *r*). На границе круга сходимости ряд может как сходиться, так и расходиться.

Для вычисления радиуса сходимости степенного ряда имеются несколько формул, например:

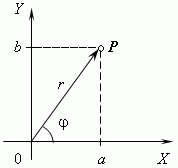
 (Формула Даламбера);

 (Формула Коши).

# Алгебра

## 10 Модуль и аргумент комплексного числа (геометрическое представление). Тригонометрическая форма записи комплексного числа. Модуль и аргумент произведения и отношения комплексных чисел.

В отличие действительных чисел, которые изображаются точками на числовой прямой, комплексные числа изображаются точками на координатной плоскости. Выберем для этого прямоугольные (декартовы) координаты с одинаковым масштабом на осях. Тогда комплексное число *a + bi* будет представлено точкой *P* с *абсцисcой a* и *ординатой b.* Эта система координат называется **комплексной плоскостью**



***Модулем*** комплексного числа называется длина вектора OP , изображающего комплексное число на координатной ( комплексной ) плоскости. Модуль комплексного числа *a + bi* обозначается | *a + bi* | или буквой *r* и равен:

*r = |a+bi| = sqrt(a^2 + b^2)*

Cопряжённые комплексные числа имеют одинаковый модуль.

***Аргумент*** *комплексного числа* - это угол  между осью *OX* и вектором *OP* , изображающим это комплексное число. Отсюда, tan  = *b* / *a* .

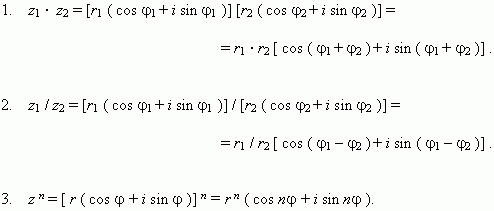
***Тригонометрическая форма комплексного числа.*** Абсциссу *a* и ординату *b* комплексного числа *a + bi* можно выразить через его модуль *r* и аргумент :

Тогда: *a = r cos* , b = r sin .

*a + bi = r (cos* + *i sin* )

***Модуль произведения двух комплексных чисел равен произведению их модулей, а аргумент — сумме их аргументов***.

***Модуль частного двух комплексных чисел равен частному модулей делимого и делителя; аргумент частного двух не равных нулю комплексных чисел равен разности аргументов делимого и делителя.***

******

## 11 Определение обратной матрицы. Теорема: необходимое и достаточное условие существования обратной матрицы (с доказательством).

Для квадратной матрицы A порядка N обратной матрицей называют квадратную матрицу B того же порядка, удовлетворяющую условию:

A \* B = B \* A = E, где E - единичная матрица

**Теорема.**

Для того, чтобы матрица A (nxn) имела обратную матрицу A-1 необходимо и достаточно, чтобы матрица A было невырожденной, то есть ее определитель не был равен 0.

**Доказательство**.

1. Существует обратная матрица A-1 , докажем, невырожденность исходной матрицы.

A \* A-1 = E

|A \* A-1| = |E|

|A| \* |A-1| = |E|

|A| \* |A-1| = 1, следовательно |A| !=0

2. Рассмотрим матрицу A~, состоящую из алгебраических дополнений элементов матрицы AT .

(A11 A21 … An1)

A~ = ( … … … ) - союзная (присоединенная матрица)

(A1n A2n … Ann)

Тогда,

(a11 a21 … an1) (A11 A21 … An1)

A \* A~ = ( … … … ) \* ( … … … )

(a1n a2n … ann) (A1n A2n … Ann)

a11 \* A11 + a12 \*A12 + … + ann \* Ann = |A|

**A \* A~ = |A| \* E**

A \* A-1 = E

A \*  \* A~ = E

A-1 **=  \*** A~

## 

## 12 Линейный оператор: определение, примеры. Ядро и образ линейного оператора. Матрица линейного оператора.

**Определение 1.** Пусть V- линейное пространство и каждому вектору x, принадлежащему V, поставлен в соответствие вектор y, y|V. Соответствие A:xRy называется **оператором**, определенным в линейном пространстве V.

Принята также запись: y=A(x). Вектор x называется ***прообразом***, а y-*образом* при **отображении** оператором A.

**Определение 2.** Линейное отображение линейного (векторного) пространства V в себя

называется **линейный преобразование** V или **линейным оператором** на V.

**Примеры**:

1. В пространстве рассмотрим следующие действия над вектором (x,y,z):
2. Поворот вокруг прямой x=y=2z на угол ;
3. Зеркальное отражение относительно плоскости 3x-y+z=0;
4. Растяжение 3.14 раза.

Все это - примеры линейных операторов. Но отображение (x,y,z) → (x+1,y,z+2) оператором не является поскольку

**2)** В пространстве отображение ортогонального проецирования на плоскость будет линейным оператором (а плоскость – не будет!). Вообще, в произвольном пространстве разбитой в прямую сумму нетривиальных пространств отображение, сопоставляет вектору Х его проекция на подпространство параллельно подпространству , будет оператором.

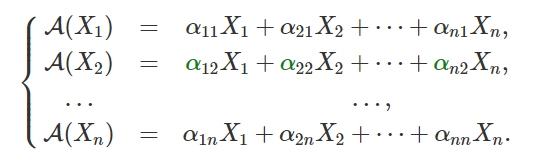
**Ядром** оператора называется множество векторов, отображаемых оператором в нулевой вектор:

V|A(X)=0};

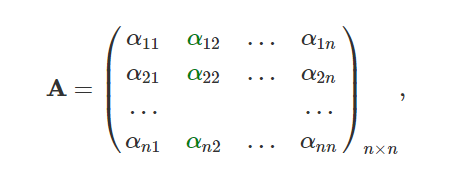
а **образом** оператора называется множество всех векторов из V, для каждого из которых существует прообраз в том же пространстве:

Рассмотрим оператор А на V базис V. Являясь частным случаем линейного отображения, оператор должен обладать и соответствующей матрицей. Существенной особенностью является невозможность произвола при выборе базиса для . Поскольку является подпространством V, то было бы слишком большой роскошью иметь два разных базиса для одного и того же пространства.

Найдем координаты образов базисных векторов в том же базисе :



Матрица



в столбцах которой стоят координаты образов базисных векторов, называется **матрицей оператора** А в базисе .

## 13 Собственные векторы и собственные значения линейного оператора. Теорема (с доказательством): необходимое и достаточное условие, чтобы действительное число являлось собственным значением линейного оператора.

Ненулевой в мне-ве пространст L - собственный вектор линейного оператора А:L→L, если : A= , а - собственное значение A

Теорема

Для того, чтобы являлось собственным значением линейного оператора необходимо и достаточно, чтобы оно являлось корнем характеристического уравнения линейного оператора.

Доказательство

Необходимость:

Пусть вектор x — собственный вектор оператора А, соответствующий собственному значению λ , т.е. по определению

; А(x) = λ(x) ; А(x) - λ(x) = 0 ; (А− λ Е) x = θ.

Следовательно, чтобы найти собственные значения и собственные векторы оператора А, нужно решить однородную систему n линейных уравнений с n неизвестными

(A − λE)X = O ненулевое решение однородной системы уравнений.

Пусть опр системы не равен 0, тогда …., решение = 0

Чтобы было ненулевое решение, |А− λ Е| = θ чтд

Достаточность:

λ - корень, |А− λ Е| = θ

Если опр равен 0 и мы выпишем систему (A − λE)X = O, то система будет иметь ненулевое решение 0 , для которого выполняется:А(x) = λ(x) и λ - собственное значение. Для нахождения x решим (A − λE)X = O

## 14 Многочлены от одной переменной. Действия над многочленами. Кольцо многочленов. Теорема о делении многочленов с остатком (формулировка). Алгоритм Евклида нахождения наибольшего общего делителя двух многочленов (с примером).

О. Для коммутативного кольца с единицей R кольцо S, состоящее из формальных сумм , где ,

Сложение задано правилом:

Умножение:

обозначается через R[x] и называется кольцом многочленов над R от одной переменной x, а элементы кольца называется многочленами.

Это коммутативное ассоциативное кольцо с единицей.

Элементы - коэффициенты многочлена

Многочлен нулевой, если все коэффициенты равны нулю.

Коэффициент многочлена при нулевой степени называется свободным членом.

Если , то он называется старшим коэффициентом, а n - степень многочлена -

Теорема о делении с остатком:

Пусть R — целостное кольцо и g — многочлен в R[x] со старшим коэффициентом, обратимым в R. Тогда каждому многочлену f ∈ R[x] сопоставляется одна и только одна пара многочленов q, r ∈ R[x], для которых:

/\* хз надо ли это

Алгоритм деления с остатком в F[X ] делает естественным рассмотрение целостного кольца R, в котором каждому элементу поставлено в соответствие неотрицательное целое числоδ(a), то есть определено отображение

δ : R \ {0} → N ∪ {0}

так, что при этом выполняются условия:

(1) δ(ab) > δ(a) для всех a, из R;

(2) каковы бы ни были a, b ∈ R, , найдутся q, r ∈ R такие,

что

a = bq + r, δ(r) < δ(q) или r = 0

Целостное кольцо R с этими свойствами называют евклидовым кольцом. Полагая δ(a) = deg a, для a = a(X) ∈ F [X], мы приходим к выводу, что F[X] евклидово кольцо.

\*/

В евклидовых кольцах существует способ нахождения НОД(a, b), называемый алгоритмом Евклида:

Пусть даны ненулевые элементы a, b евклидова кольца R. Применяя достаточно большое (но конечное) число раз правило (2), мы получим систему равенств с последним нулевым остатком:

. . . . . . . . . . . . . . . . . . . . . . . . . . . . .

Последний отличный от нуля остаток является как раз наибольшим общим делителем элементов a и b.

ПРИМЕР:

a = , b =

+ 1

5 = 5\*1 + 0

НОД(a,b) = 1

# Алгоритмы вычислительной математики

## 15 Погрешность результата численного решения задач. Источник и классификация погрешности. Предельная абсолютная и относительная погрешности. Погрешность арифметических действий над приближенными числами.

Погрешности решения задач обусловливаются следующими причинами: 1. Математическое описание задач (математическая модель) большей частью является неточным;

2. Методы решения задач не являются точными. Во многих случаях получение точного решения требует выполнения неограниченного количества шагов. Обрыв бесконечного процесса приводит к получению приближенного решения

3. Исходные данные для решения задач, как правило, получаются из эксперимента, а каждый эксперимент может дать результат с ограниченной точностью;

4. При вводе исходных данных в машину, при выполнении арифметических операций, при выводе информации производятся округления;

5. Погрешности приближенных чисел (погрешности исходных данных и погрешности округления) в процессе решения задачи будут последовательно переходить (чаще всего в увеличенном размере) в результаты вычислений и порождать новые погрешности.

В соответствии с указанными источниками погрешностей можно осуществить классификацию последних:

А) Неустранимые погрешности:

1) математического описания задачи [1];

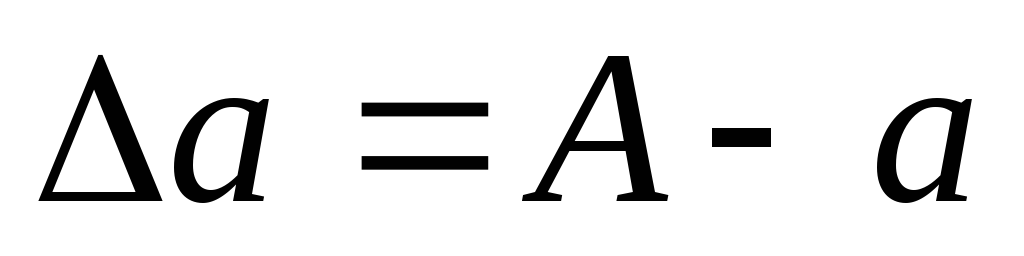
2) исходных данных [3];

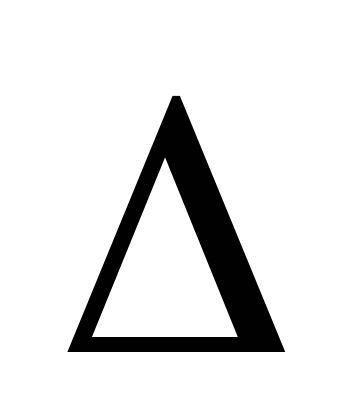
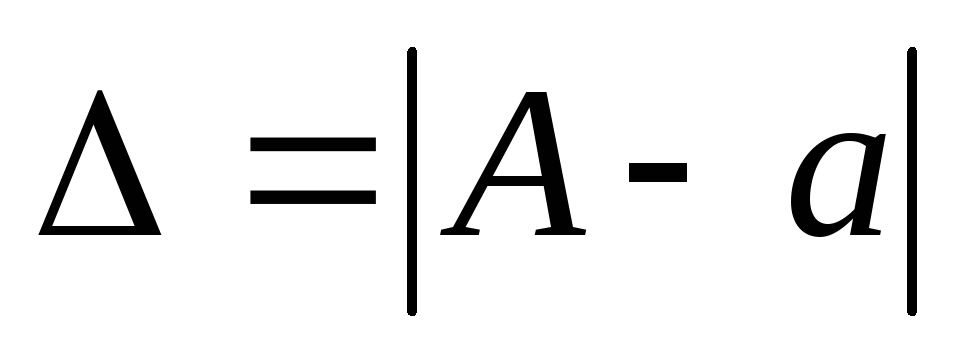
Б) погрешности метода [2];

В) вычислительные погрешности [4,5].

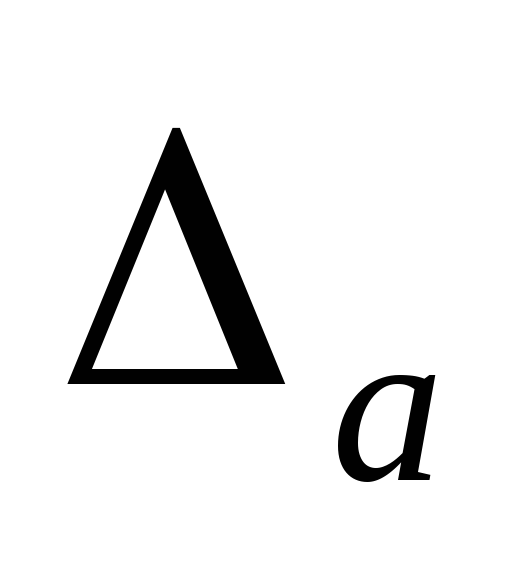
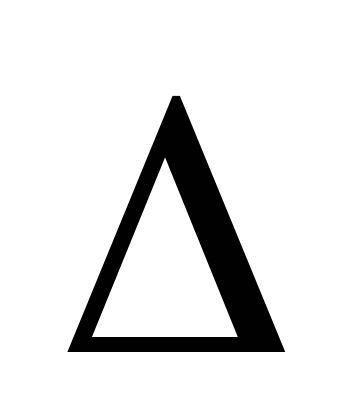
В качестве характеристик точности приближенных величин любого происхождения вводятся понятия абсолютной и относительной погрешности этих величин.

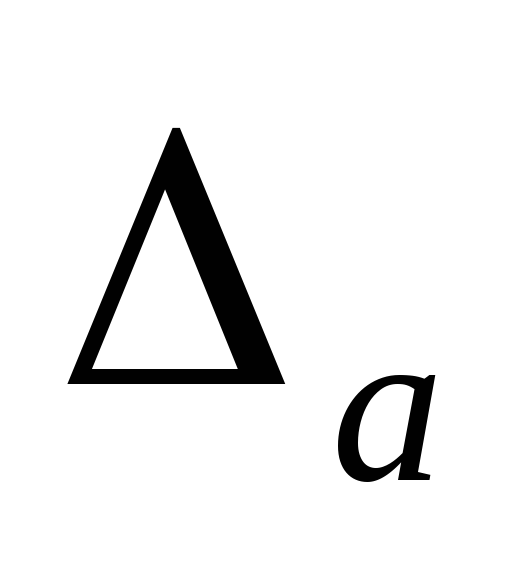
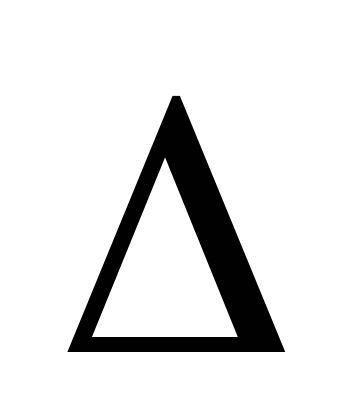
Обозначим через а приближение к точному числу А.

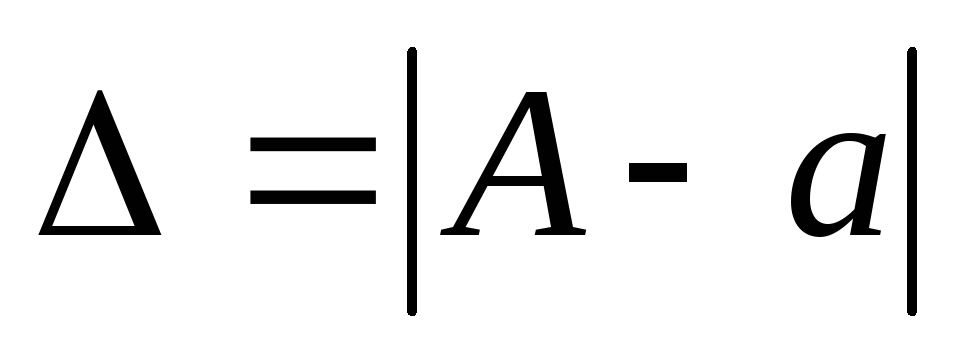
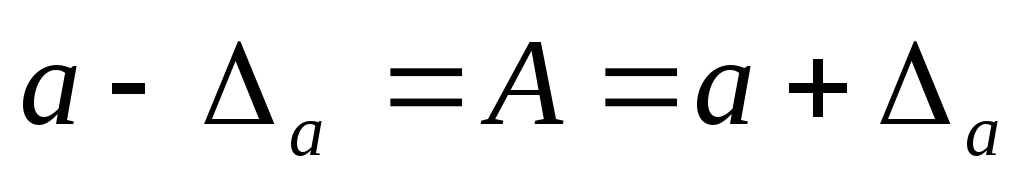
**Определени.** Величина называется погрешностью приближенного числа а.

**Определение.** Абсолютной погрешностью  приближенного числа а называется величина .

Практически точное число А обычно неизвестно, но мы всегда можем указать границы, в которых изменяется абсолютная погрешность.

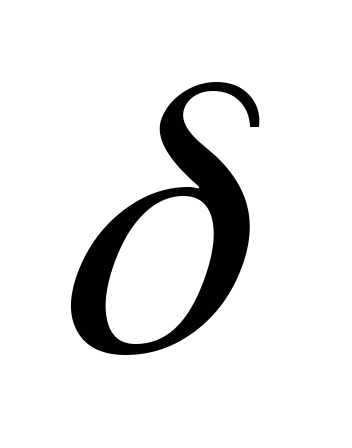
**Определение.** Предельной абсолютной погрешностью  приближенного числа а называется наименьшая из верхних границ для величины , которую можно найти при данном способе получения числа а.

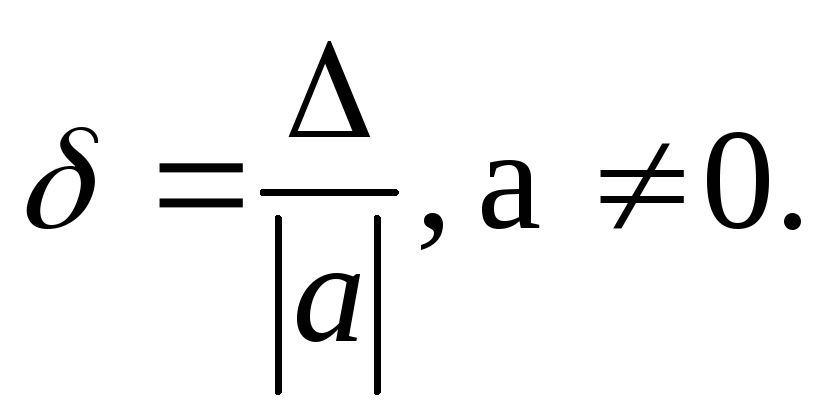
На практике в качестве  выбирают одну из верхних границ для , достаточно близкую к наименьшей.

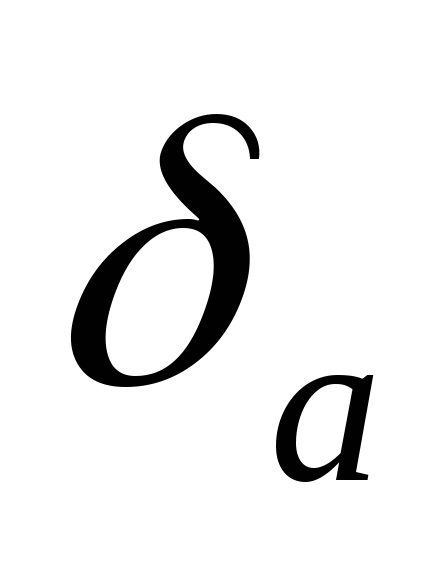
Поскольку , то. Иногда пишут:.

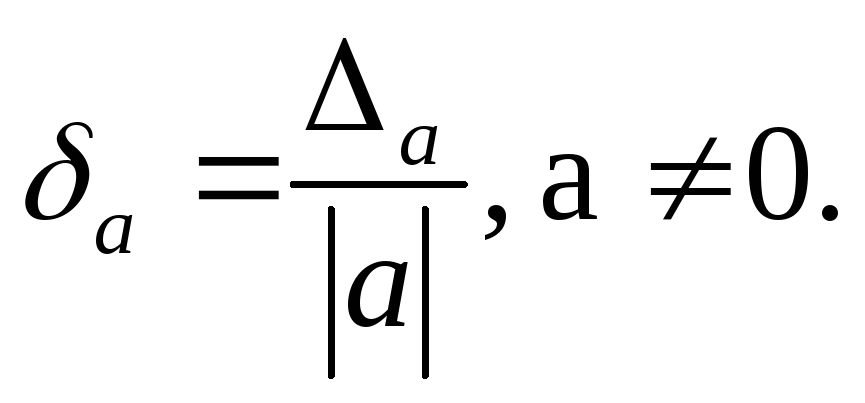
**Абсолютная погрешность** - это разница между результатом измерения и истинным (действительным) значениемизмеряемой величины.

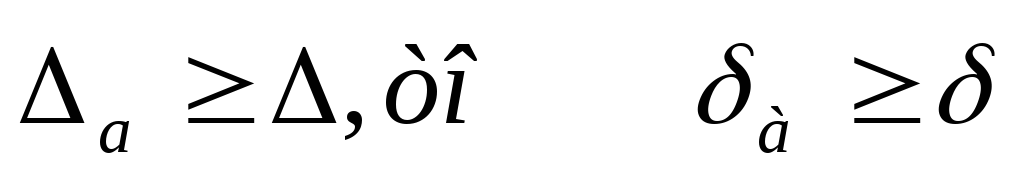
Абсолютная погрешность и предельная абсолютная погрешность не достаточны для характеристики точности измерения или вычисления. Качественно более существенна величина относительной погрешности.

**Определение.** Относительной погрешностью  приближенного числа а назовем величину:



**Определение.** Предельной относительной погрешностью  приближенного числа а назовем величину



Так как 

Предельная абсолютная погрешность суммы приближенных чисел равна:

Δu = Δx + Δy

Если все слагаемые в сумме имеют один и тот же знак, то предельная относительная погрешность суммы не превышает наибольшей из предельных относительных погрешностей слагаемых:

δu <= max(δx1, δx2, … δxn)

При вычислении разности двух приближенных чисел u = х - у ее абсолютная погрешность равна Δu = Δx + Δy, а предельная относительная погрешность:

δu= (Δx + Δy) / |x - y|

Из формулы следует, что если приближенные значения х и у близки, то предельная относительная погрешность будет очень большой.

В некоторых случаях удается избежать вычисления разности близких чисел с помощью преобразования выражения так, чтобы разность была исключена.

Если представляется сложным заменить вычитание близких приближенных чисел сложением, то следует поступать так: если известно, что при вычитании должно пропасть m первых значащих цифр, а в результате требуется сохранить n верных цифр, тогда в уменьшаемом и вычитаемом следует сохранять m + n верных значащих цифр:

sqrt(4,05) - sqrt(4) ≈ 2,012461 - 2 = 0.0125

Предельная относительная погрешность произведения u = х ∙у приближенных чисел, отличных от нуля, равна:

δu = δx + δy

В частности, если u = kx, где k — точное число, имеем:

Δu = |k| Δx

δu = δx

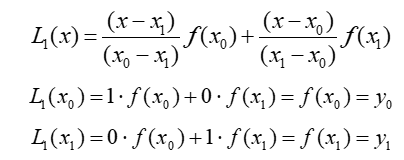
Предельная относительная погрешность частного равна:

δu = δx + δy

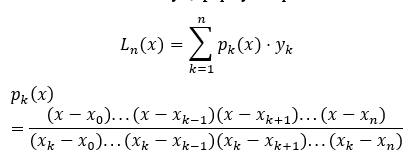
## 16 Постановка задачи интерполяции функции. Интерполяционный полином Лагранжа. Погрешность интерполяции для равноотстоящих узлов.

**Постановка задачи.** Пусть на [a,b] заданы n+1 точка x0,...xn и значения функции в этих точках y0 = f(x0)...yn=f(xn) . Требуется построить полином степени не выше n, имеющий в заданных точках те же значения, что и функция f(x).

Пусть заданы две точки (x0,y0) и (x1,y1). Тогда интерполяционный полином Лагранжа вычисляется по следующим формулам:



Если точек больше двух, формула принимает вид:



**Теорема.** Интерполяционный полином Лагранжа существует и определяется единственным образом.

Доказательством теоремы служит построение данного полинома, удовлетворяющего условиям.

Единственность:

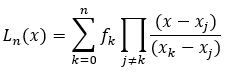
Предположим противное. Построим полином n-го порядка Ln(x) и Ln`(x), Ln(x) != Ln`(x) .

Рассмотрим разность Ln(x) - Ln`(x) = p(x), степень(p(x))<n.

Найдем значения разности в точках xi, i = 0,...n. Эти значения равны нулю, следовательно, полином максимум n-й степени имеет n+1 корень – противоречие.

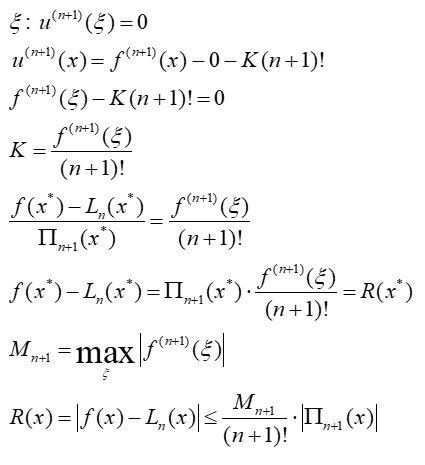
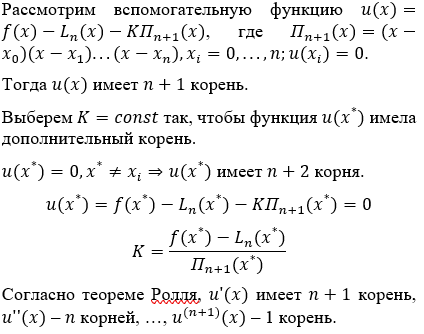
**Погрешность интерполяции**

Пусть f(x) задана аналитически.



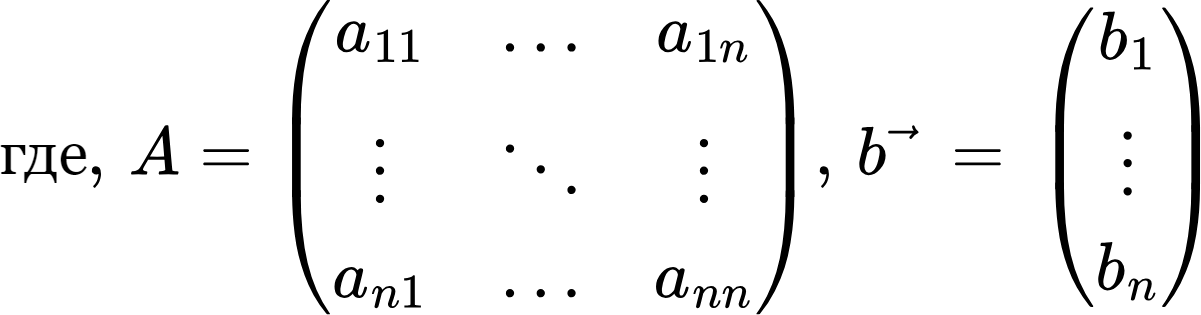
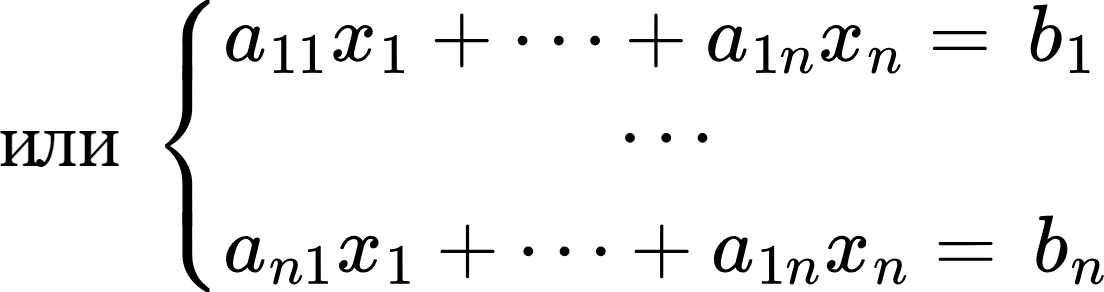
Посчитать погрешность R(x)=f(x)-Ln(x), в точках x, не являющихся точками интерполяции.

Пусть f(x) гладкая функция и имеет производные до n+1 порядка.

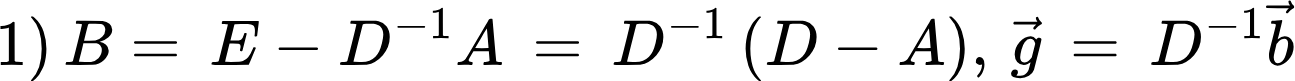


## 17 Итерационные метод Якоби (простой итерации) решения систем линейных алгебраических уравнений. Достаточные условия сходимости метода.

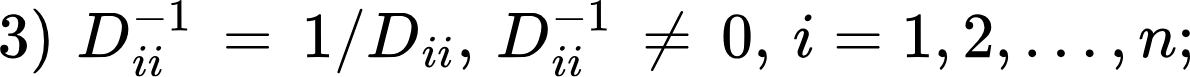
Возьмем систему: 𝐴𝑥⃗=𝑏⃗,

Метод Якоби – разновидность метода простой итерации для решения системы линейных алгебраических уравнений. Назван в честь Карла Густава Якоби. Для того, чтобы построить итеративную процедуру метода Якоби, необходимо провести предварительное преобразование системы уравнений 𝐴𝑥⃗ = 𝑏⃗⃗ к итерационному виду 𝑥⃗ = 𝐵𝑥⃗ + 𝑔⃗. Оно может быть осуществлено по одному из следующих правил:

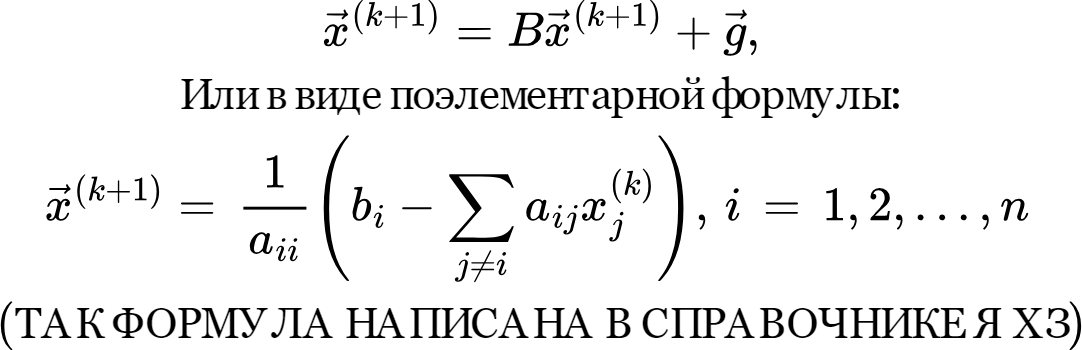
;

{"backgroundColor":"#ffffff","font":{"size":12,"color":"#000000","family":"Times New Roman"},"aid":null,"id":"4","code":"$$2)\\,B\\,=\\,-D^{-1}\\left(L+U\\right)=\\,-D^{-1}\\left(A-D\\right),\\,\\vec{g}=D^{-1}\\vec{b};$$","type":"$$","ts":1716234043619,"cs":"36g/YSg727BH6ILmp5BGBA==","size":{"width":360,"height":20}}

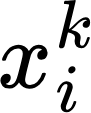
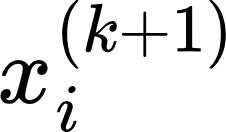


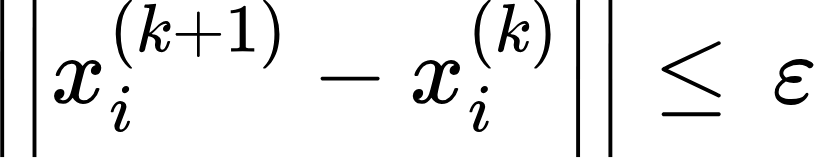
где в принятых обозначениях 𝐷 означает матрицу, у которой на главной диагонали стоят соответствующие элементы матрицы 𝐴, а все остальные нули; тогда как матрицы 𝑈 и 𝐿 содержат верхнюю и нижнюю треугольные части 𝐴, на главной диагонали которых нули, 𝐸 – единичная матрица.

Тогда процедура нахождения решения имеет вид:

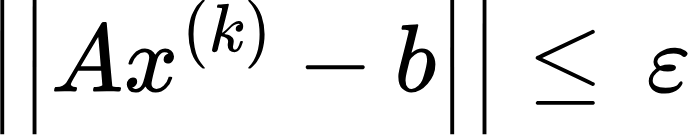


, где k - счетчик итерации

В отличие от метода Гаусса-Зейделя нельзя заменять  на  в процессе итерационной процедуры, так как эти значения понадобятся для остальных вычислений. Это наиболее значимое различие между методом Якоби и методом Гаусса-Зейделя решения СЛАУ. Таким образом на каждой итерации придётся хранить оба вектора приближений: старый и новый.

Условие окончания итерационного процесса при достижении точности 𝜀 имеет вид:

Данный критерий не требует вычисления нормы матрицы и часто используется на практике. При этом точное условие окончания итерационного процесса имеет вид



и требует дополнительного умножения матрицы на вектор на каждой итерации, что примерно в два раза увеличивает вычислительную сложность алгоритма.

На плюсах

const double eps = 0.001; ///< желаемая точность

/// N - размерность матрицы; A[N][N] - матрица коэффициентов, F[N] - столбец свободных членов,

/// X[N] - начальное приближение, ответ записывается также в X[N];

do {

for (int i = 0; i < N; i++) {

TempX[i] = F[i];

for (int g = 0; g < N; g++) {

if (i != g)

TempX[i] -= A[i][g] \* X[g];

}

TempX[i] /= A[i][i];

}

norm = fabs(X[0] - TempX[0]);

for (int h = 0; h < N; h++) {

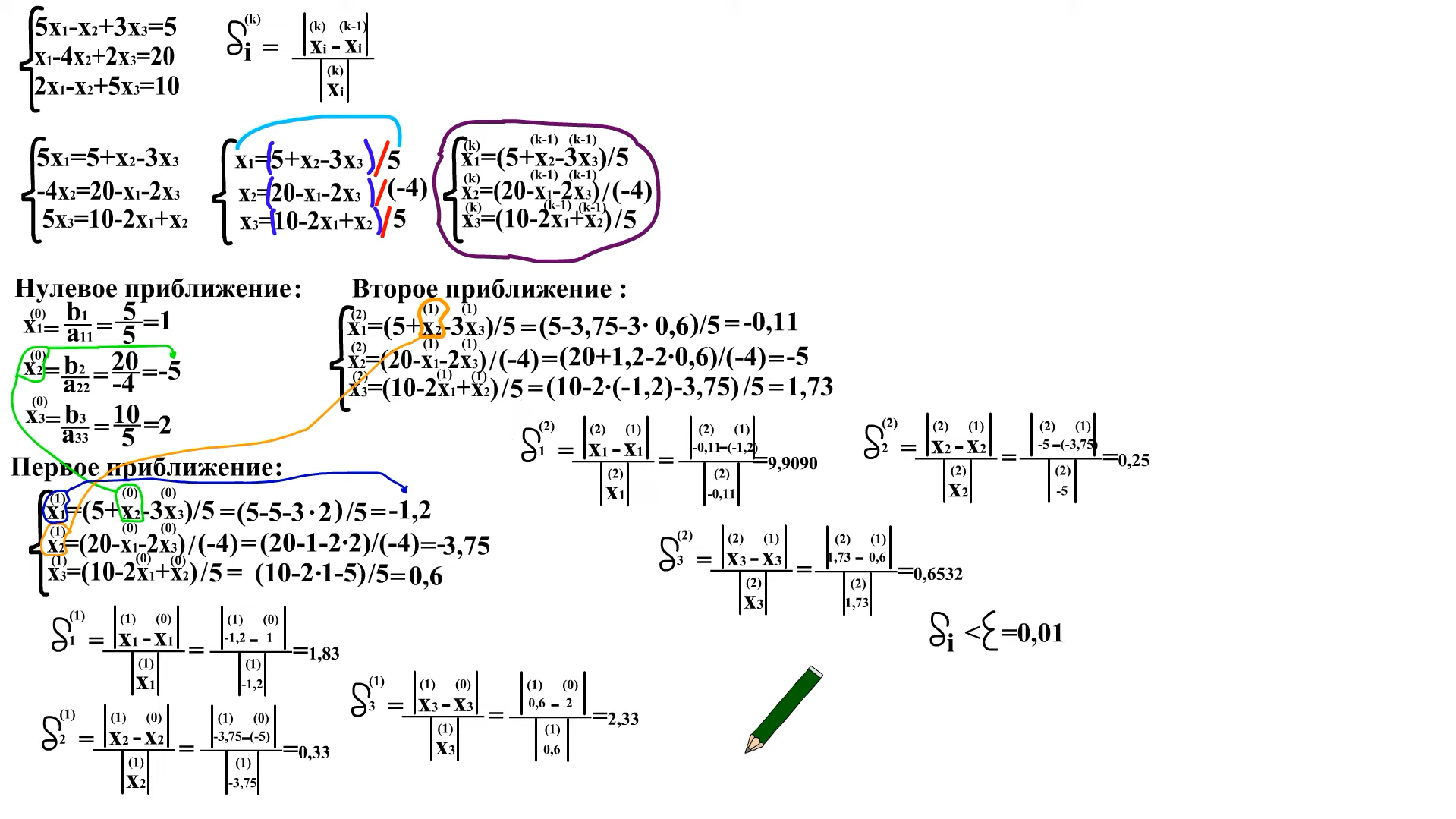
if (fabs(X[h] - TempX[h]) > norm)

norm = fabs(X[h] - TempX[h]);

X[h] = TempX[h];

}

} while (norm > eps);

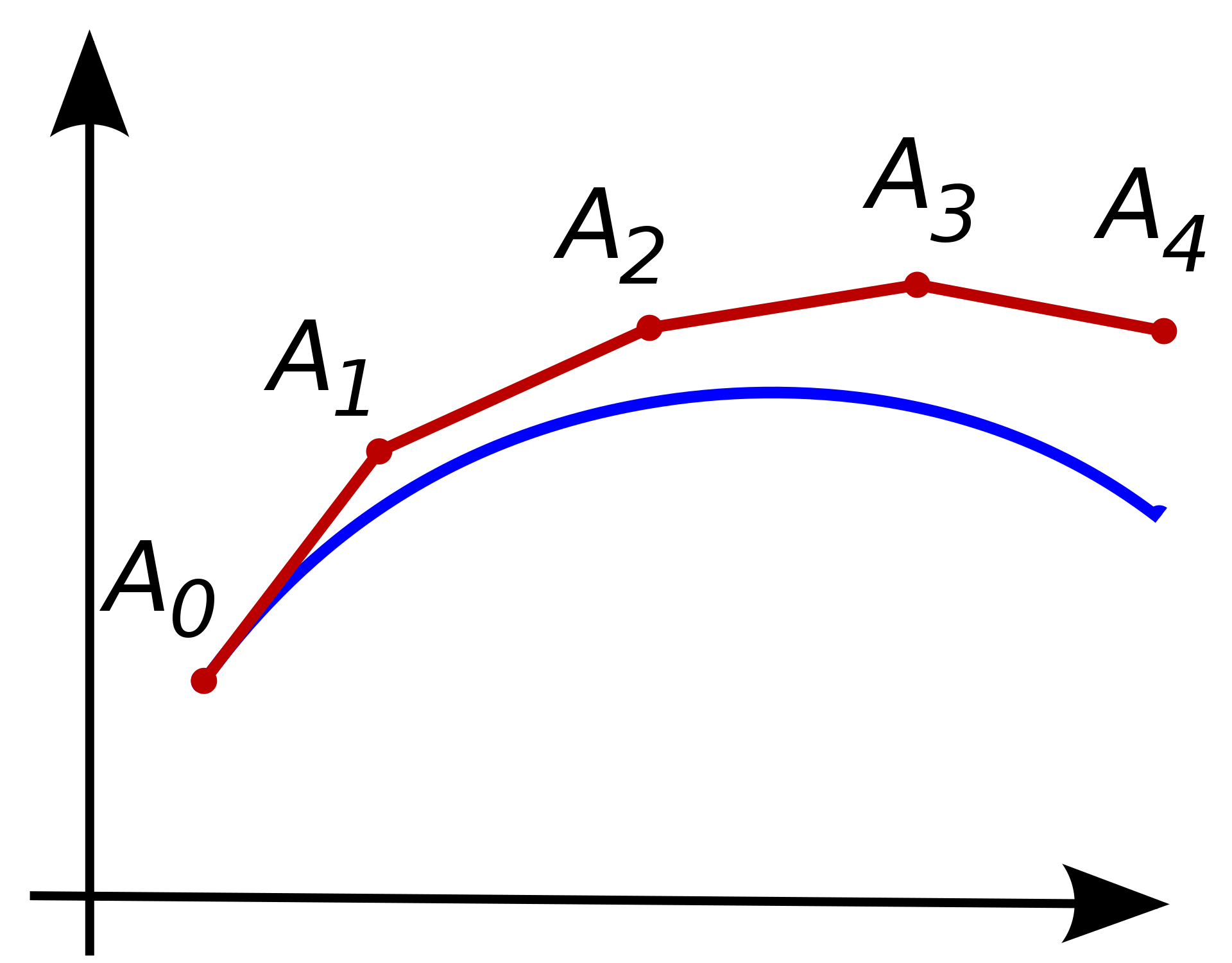


Считаем приближения пока у нас не будет минимальная погрешность

## 18 Метод Эйлера, решения задачи Коши для обыкновенных дифференциальных уравнений, вывод метода, оценка погрешности. Геометрическая интерпретация метода Эйлера.

В основе метода Эйлера лежит идея графического решения дифференциального уравнения:

Уравнение вместе с начальными условиями задают направление касательной к искомой интегральной кривой.



Уравнение касательной:

Разложим искомое решение в ряд Тейлора:

Положим , тогда

Погрешность при первом шаге:

Погрешность при многих шагах:

где - константа Липшица,

# 

# Основы нечеткой математики

## 19 Понятие нечеткого множества. Способы задания нечеткого множества. Меры нечеткости множества.

Нечетким множеством А называют множество упорядоченных пар

A={ где - функция, определенная на универсальном множестве U и принимающая значения на отрезке [0, 1].

Функция - характеристическая функция или функция принадлежности множества А. Функция принадлежности (membership function) позволяет вычислить степень принадлежности произвольного элемента универсального множества к нечеткому множеству.

Способы задания: графический, математический (через функции принадлежности), лингвистический (с помощью лингвистических переменных).

Мерой нечеткости множества А является функция d(A), которая множеству А ставит в соответствие действительное число и удовлетворяет следующим аксиомам меры нечеткости:

d(A)=0 А – обычное множество;

d(A) принимает максимальное значение - = 0,5

d(A) <= d(B) если:

Если = 1- , то d(A)=d(B)

Линейное расстояние Хэмминга:

ρ(,

ρ(,

Евклидово расстояние:

ρ(,

ρ(,

## 20 Нечеткие бинарные отношения: понятие, способы задания. Композиция нечетких бинарных отношений.

Содержательно нечеткое отношение определяется как любое нечеткое подмножество упорядоченных кортежей, построенных из элементов тех или иных базисных множеств, в качестве которых в данном случае используются универсумы. При этом под кортежем, так же как и в случае обычных множеств, понимается произвольный набор или список упорядоченных элементов.

**Определение.** В общем случае нечетким отношением или, более точно, нечетким k-арным отношением, заданным на множествах (универсумах) называется некоторое фиксированное нечеткое подмножество декартова произведения этих универсумов. Другими словами, если обозначить произвольное нечеткое отношение через , то по определению },

где — функция принадлежности данного нечеткого отношения, которая определяется как отображение .

Здесь через обозначен кортеж из k элементов, каждый из которых выбирается из своего универсума. Так же как и в случае обычных множеств с целью характеризовать количество универсальных множеств, на основе которых строится то или иное нечеткое отношение, принято называть нечеткое отношение между элементами из двух универсальных множеств — бинарным, между элементами трех множеств — тернарным, а в общем случае— *k-арным* отношением. При этом на форму и вид функции принадлежности нечеткого отношения предварительно не накладывается никаких ограничений.

**Определение.** В теории нечетких отношений пустое нечеткое отношение определяется как отношение, которое не содержит ни одного кортежа. Это отношение по-прежнему обозначается через Ø и формально определяется как такое нечеткое отношение, функция принадлежности которого тождественно равна 0 на всем декартовом произведении его универсумов. Из этого определения также следует, что пустое нечеткое отношение совпадает с обычным пустым отношением.

**Определение.** Что касается другого крайнего случая, то так называемое полное нечеткое отношение по своей сути совпадает с обычным полным отношением, которое, в свою очередь, равно по определению декартову произведению соответствующих универсумов . Как его обозначать — из соображений удобства можно просто через X. Важно представлять себе, что функция принадлежности полного нечеткого отношения тождественно равна единице для всех без исключения кортежей, т. е. . Особое значение в нечетком моделировании имеют бинарные нечеткие отношения, для задания которых используется одно или разные базисные множества (универсумы). В связи с этим приведем их формальное определение.

**Определение.** *В общем случае бинарное нечеткое отношение задается на базисных множествах и определяется как нечеткое отношение* }. Здесь — функция принадлежности бинарного нечеткого отношения, которая определяется как отображение :, а через обозначен кортеж из двух элементов, при этом , .

**Определение.** Применительно к бинарным нечетким отношениям определяется так называемое обратное нечеткое отношение. А именно, если задано бинарное нечеткое отношение на декартовом произведении , то обратным к нему нечетким отношением (обозначается через ]) называется такое бинарное нечеткое отношение, которое задано на декартовом произведении , а функция принадлежности которого определяется по следующей формуле: = .

**Определение.** Бинарное нечеткое отношение, заданное на одном базисном множестве (универсуме) X, определяется как нечеткое отношение , } где — функция принадлежности бинарного нечеткого отношения, которая определяется как отображение :.

**Способы задания бинарных нечетких отношений.**

1. В форме списка с явным перечислением всех кортежей нечеткого отношения и соответствующих им значений функции принадлежности
2. Аналитически в форме некоторого математического выражения для соответствующей функции принадлежности этого нечеткого отношения. Этот способ может быть использован для задания произвольных нечетких отношений как с конечным, так и с бесконечным числом кортежей.
3. Графически в форме некоторой поверхности или совокупности отдельных точек в трехмерном пространстве. При этом две координаты (независимые переменные) будут соответствовать значениям универсумов и , а третья координата — интервалу [0,1]. Например, график математической функции: для может служить примером графического способа формального задания некоторого нечеткого отношения. Здесь функция z является представлением функции принадлежности соответствующего нечеткого отношения. Этот способ зачастую используется в дополнение к аналитическому для визуализации бинарных нечетких отношений с бесконечным числом кортежей.
4. В форме матрицы нечеткого отношения. Этот способ основан на представлении нечеткого бинарного отношения с конечным числом кортежей в форме матрицы , строки которой соответствуют первым элементам кортежей, а столбцы — вторым элементам кортежей рассматриваемого нечеткого отношения. При этом элементами матрицы являются соответствующие значения функции принадлежности данного отношения. Если бинарное нечеткое отношение задается на одном универсуме, то матрица такого отношения является квадратной. Определенную таким образом матрицу называют матрицей бинарного нечеткого отношения и обозначают .
5. В форме так называемого нечеткого графа, который формально может быть задан в виде двух обычных конечных множеств и некоторой функции принадлежности. А именно, нечеткий граф, а точнее, ориентированный нечеткий граф, есть , где — множество вершин нечеткого графа, — множество дуг нечеткого графа, — функция принадлежности дуг данному нечеткому графу, т. е. . При этом вершины нечеткого графа, как и в случае обычных графов, изображаются точками, дуги — отрезками прямых линий со стрелкой на одном из концов. Рядом с вершинами записываются условные обозначения соответствующих вершин, а рядом с каждой дугой — значение функции принадлежности для соответствующей дуги. Натуральное число n определяет общее количество вершин конкретного нечеткого графа, а натуральное число m — общее количество дуг нечеткого графа. При этом дуги с нулевой функцией принадлежности в нечетком графе обычно не изображаются.

**Композиция бинарных нечетких отношений.**

Пусть и R — конечные или бесконечные бинарные нечеткие отношения. Причем нечеткое отношение } задано на декартовом произведении универсумов , а нечеткое отношение } — на декартовом произведении универсумов .

Нечеткое бинарное отношение, заданное на декартовом произведении и обозначаемое через , называется композицией бинарных нечетких отношений и R, а его функция принадлежности определяется следующим выражением: . Определенную таким образом композицию бинарных нечетких отношений называют иногда (max-min)-композицией или максиминной сверткой нечетких отношений.

## 21 Нечеткие булевы переменные и логические операции над ними. Функции нечетких булевых переменных: определение, составление таблиц функций.

**Нечеткой булевой переменной** называют переменную р, которая является именем нечеткого подмножества множества U = [0,1].

Правила выполнения логических операций над нечеткими булевыми переменными:

Не выполняются законы:

Выполняются свойства порядка:

Функцию называют **функцией нечетких булевых переменных,** если она принимает значения на [0,1].

**Пример построения таблицы функций:**

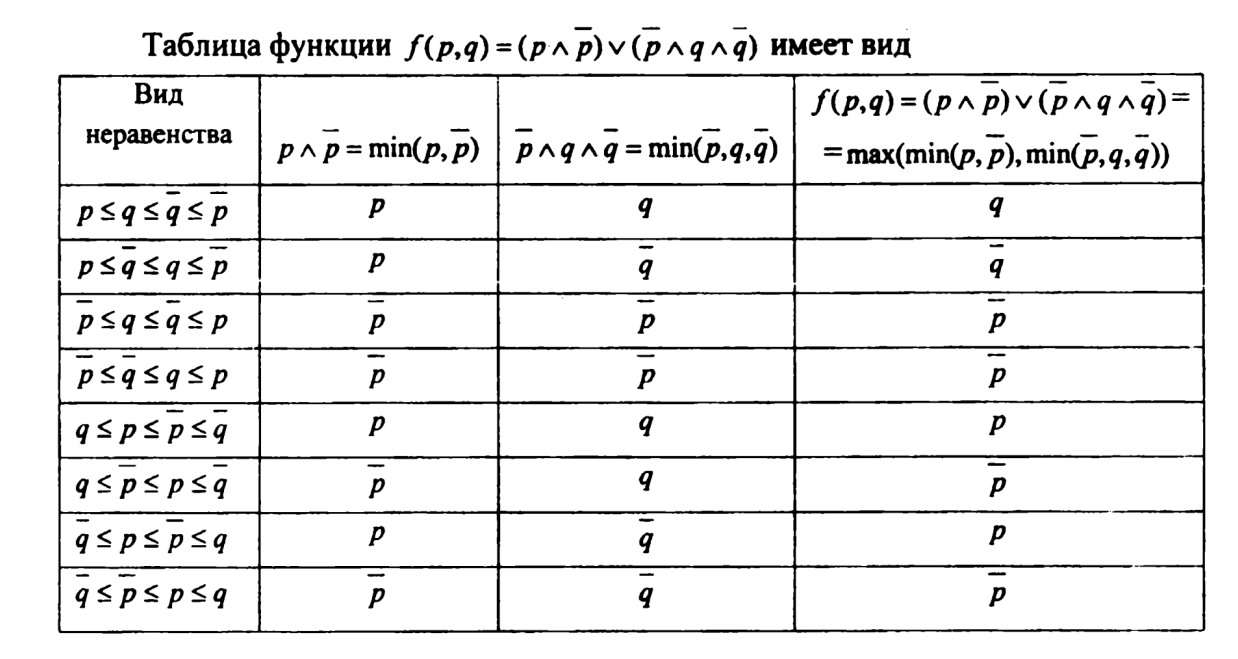
, переменные p,q, ,

Не может быть неравенст вида p ≤ q ≤ ≤

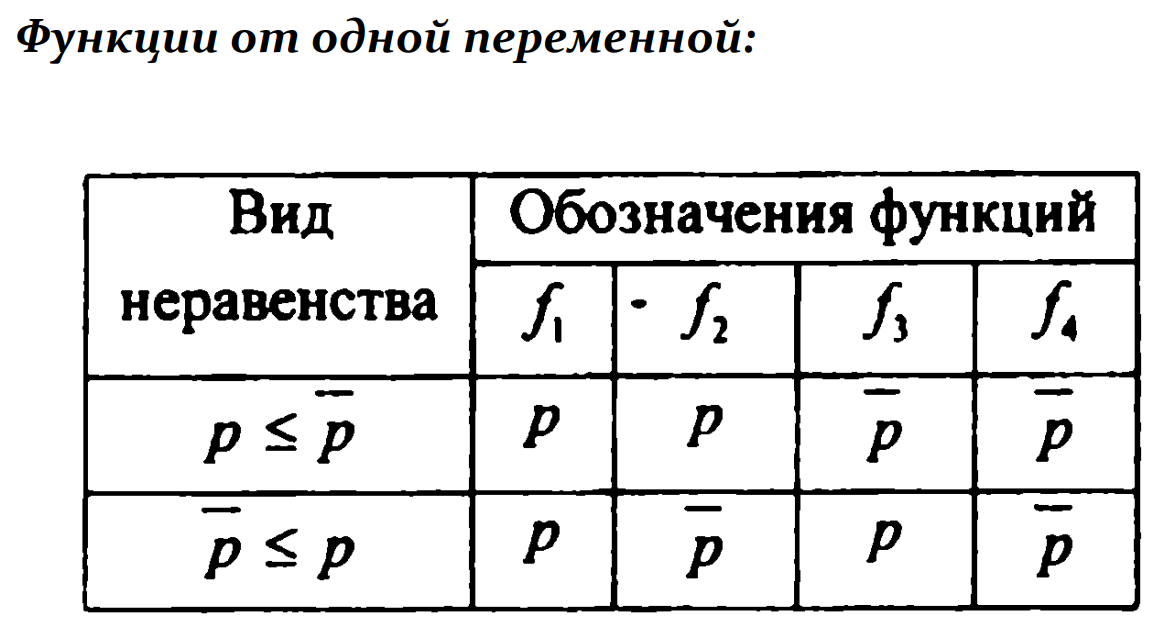
p=0.3, q=0.4

Рассматриваем 8 неравенств:

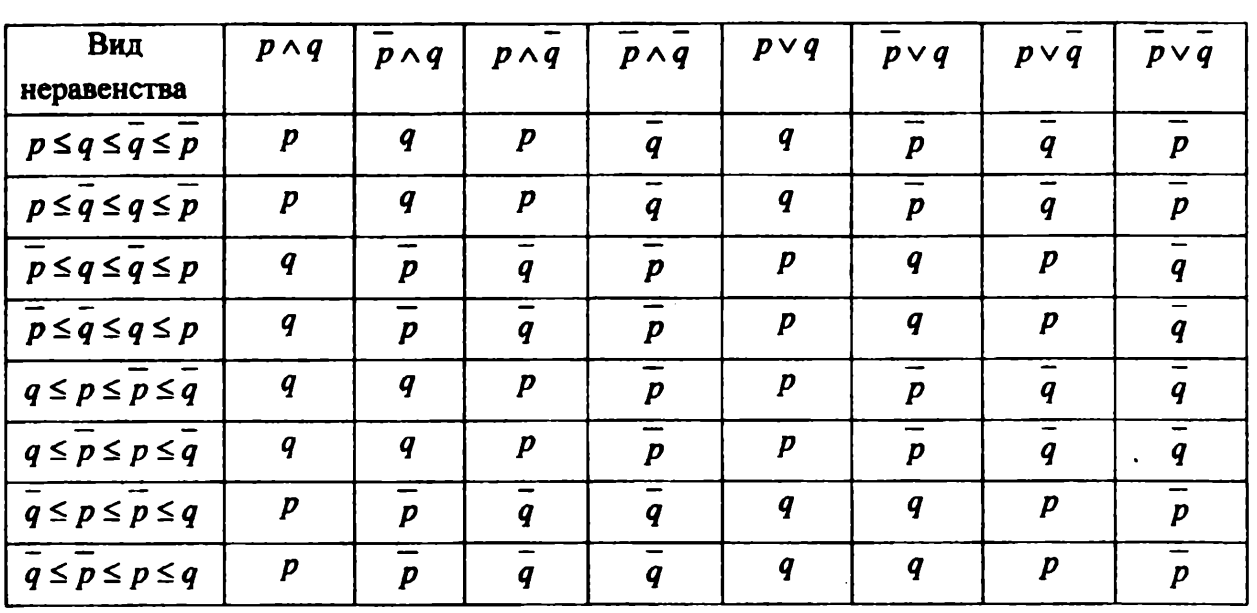
)



**ДОП ИНФА ПРО ЭТИ ТАБЛИЦЫ(мб вам пригодится)**



И от двух переменных:



# Дифференциальные и разностные уравнения

## 22 Теорема существования и единственности решения задачи Коши для линейных систем дифференциальных уравнений. Свойства решений линейных однородных систем дифференциальных уравнений. Теорема о представлении общего решения (с доказательством).

Теорема существования и единственности решения задачи Коши для линейных систем дифференциальных уравнений

**Теорема**. Рассмотрим линейную систему дифференциальных уравнений:

𝑦′=𝐴(𝑡)𝑦+𝑓(𝑡) , где 𝑦=𝑦(𝑡)— вектор неизвестных функций, 𝐴(𝑡) — матрица коэффициентов размером 𝑛×𝑛, и 𝑓(𝑡) — вектор внешних воздействий (вектор-столбец размером 𝑛×1). Пусть 𝐴(𝑡) и 𝑓(𝑡) непрерывны на некотором промежутке 𝐼⊂𝑅, содержащем 𝑡0​ . Тогда для любой начальной точки 𝑦(𝑡0)=𝑦0 существует единственное решение задачи Коши на этом промежутке.

**Доказательство** (схематичное).

Для доказательства теоремы применим метод Пикара-Линделёфа. Решение будем искать в виде ряда Пикара: 𝑦(𝑡)=𝑦0+∫𝑡0𝑡𝐴(𝑠)𝑦(𝑠)+𝑓(𝑠) 𝑑𝑠.

Итерационный процесс Пикара: 𝑦𝑛+1(𝑡)=𝑦0+∫𝑡0𝑡𝐴(𝑠)𝑦𝑛(𝑠)+𝑓(𝑠) 𝑑𝑠. Этот процесс сходится при достаточной малости ∣𝑡−𝑡0∣. Предел этого процесса и будет искомым решением.

**Свойства решений линейных однородных систем дифференциальных уравнений**

Рассмотрим однородную систему: 𝑦′=𝐴(𝑡)𝑦.

1. Линейность решения. Если 𝑦1(𝑡) и 𝑦2(𝑡) — решения однородной системы, то любое их линейное сочетание 𝑐1𝑦1(𝑡)+𝑐2𝑦2(𝑡) также является решением.

2. Существование фундаментальной матрицы решений. Для однородной системы существует фундаментальная матрица решений Φ(𝑡), такая что любой вектор-столбец решений может быть представлен в виде 𝑦(𝑡)=Φ(𝑡)𝑐, где 𝑐 — вектор произвольных постоянных.

3. Свойства фундаментальной матрицы.

Фундаментальная матрица Φ(𝑡) обладает свойством Φ′(𝑡)=𝐴(𝑡)Φ(𝑡). Если Φ(𝑡) — фундаментальная матрица, то Φ(𝑡)Φ^−1(𝑡0) = 𝐼, где 𝐼 — единичная матрица.

**Теорема о представлении общего решения (с доказательством**)

Теорема. Общее решение неоднородной системы линейных дифференциальных уравнений 𝑦′=𝐴(𝑡)𝑦+𝑓(𝑡) может быть представлено в виде 𝑦(𝑡)=Φ(𝑡)𝑐+𝑦𝑝(𝑡), где Φ(𝑡) — фундаментальная матрица решений соответствующей однородной системы, 𝑐 — вектор произвольных постоянных, 𝑦𝑝(𝑡) — частное решение неоднородной системы.

**Доказательство**.

Нахождение фундаментальной матрицы. Пусть Φ(𝑡) — фундаментальная матрица решений однородной системы 𝑦′=𝐴(𝑡)𝑦.

Нахождение частного решения. Для нахождения частного решения используем метод вариации произвольных постоянных. Предположим, что 𝑦𝑝(𝑡) имеет вид 𝑦𝑝(𝑡)=Φ(𝑡)𝑢(𝑡), где 𝑢(𝑡) — вектор-функция.

Определение 𝑢(𝑡). Подставим 𝑦𝑝(𝑡)=Φ(𝑡)𝑢(𝑡) в неоднородное уравнение: Φ′(𝑡)𝑢(𝑡)+Φ(𝑡)𝑢′(𝑡)=𝐴(𝑡)Φ(𝑡)𝑢(𝑡)+𝑓(𝑡).

Поскольку Φ(𝑡) — фундаментальная матрица, Φ′(𝑡)=𝐴(𝑡)Φ(𝑡), уравнение упрощается до: Φ(𝑡)𝑢′(𝑡)=𝑓(𝑡). Решая это уравнение относительно 𝑢′(𝑡), получаем: 𝑢′(𝑡)=Φ−1(𝑡)𝑓(𝑡). Интегрируя, находим 𝑢(𝑡): 𝑢(𝑡)=∫Φ−1(𝑡)𝑓(𝑡) 𝑑𝑡. Таким образом, частное решение 𝑦𝑝(𝑡)=Φ(𝑡)∫Φ−1(𝑡)𝑓(𝑡) 𝑑𝑡.

Общее решение неоднородной системы имеет вид: 𝑦(𝑡)=Φ(𝑡)𝑐+Φ(𝑡)∫Φ−1(𝑡)𝑓(𝑡) 𝑑𝑡.

Сгруппировав, получаем: 𝑦(𝑡)=Φ(𝑡)(𝑐+∫Φ−1(𝑡)𝑓(𝑡) 𝑑𝑡). Таким образом, общее решение неоднородной системы состоит из общего решения однородной системы и частного решения неоднородной системы.

## 23 Теорема существования и единственности решения задачи Коши для линейных дифференциальных уравнений n-го порядка. Свойства решений линейных однородных дифференциальных уравнений n-го порядка, теорема о представлении общего решения (с доказательством).

**Теорема существования и единственности решения задачи Коши для линейных дифференциальных уравнений n-го порядка**

(2.2)

Если в дифференциальном уравнении (2.2) коэффициенты , и функция непрерывна на , то при любых начальных условиях

) = (2.3)

Существует и притом единственное в некотором промежутке решение уравнения (2.2), удовлетворяющее начальным условиям (2.3)

**Свойства решений линейных однородных дифференциальных уравнений n-го порядка:**

**Теорема (1).**

Линейная комбинация решений линейной однородной системы

Σ(k=1)( cₖ \* yᵏ(x) ) является так же решением этой системы

**Теорема (2).**

Если вектор функции y¹(x), . . . , yᵏ(x) линейно зависимы, то определитель Вронского, построенный по этим функциям тождественно равен нулю.

****

**Теорема (3).**

Пусть вектор функции y¹(x), . . . , yᵏ(x) — решения линейной однородной дифференциальной системы (1). тогда если определитель Вронского, составленный по этим вектор-функциям, равен нулю в некоторой точке x0, то эти вектор функции линейно зависимы

**Определение.** n линейно независимых решений линейной однородной системы (1) называется фундаментальной системой решений этой системы. n - размерность системы (1)

**Теорема (4).**

Фундаментальная система решений существует у каждой линейной однородной системы (1).

**Теорема (5).**

Общее решение линейной однородной системы (1) — это линейная комбинация фундаментальной системы решений с произвольными постоянными .

**Теорема о представлении общего решения (с доказательством)**

Если функции образуют ФСР линейного однородного дифференциального уравнения:

с непрерывными коэффициентами то их линейная комбинация: (17), где , i = 1,2,...n - произвольные постоянные, является общим решением уравнения

**Доказательство:**

1. Пусть в точке заданы условия: , ,..., .
2. Найдём производные функции (17): y’,y’’,..., и запишем для заданной начальной точки :

………………………………………………

(18)

1. Система (18) есть система линейных однородных уравнений. По предположению (так как функции образуют ФСР) определитель этой системы , следовательно, для точки система (18) имеет единственное ненулевое решение (. Это значит, что для функции: задача Коши решается однозначно, то есть эта функция есть ОБЩЕЕ РЕШЕНИЕ. ЧТД 🖕

## 24 Представление общего решения линейного однородного дифференциального уравнения n-го порядка с постоянными комплексными и вещественными коэффициентами. Поиск частного решения линейного неоднородного уравнения n-го порядка методом вариации произвольных постоянных (с доказательством).

**Часть 1:**

ЛОДУ n-го порядка с постоянными коэффициентами - уравнение вида

(в скобках - порядок производной)

Общее решение такого ДУ - линейная комбинация частных решений, образующих фундаментальную систему.

Для решения составляем и решаем характеристическое уравнение:

Каждому корню соответствует решение ДУ:

* , если λ - не кратный корень (встречается не кратное кол-во раз)

Если же - кратный корень кратности , то он превращается в решений ДУ:

Комплексно-сопряженной паре корней соответствует решение:

**Часть 2:**

ЛНДУ n-го порядка - уравнения вида

(в скобках - порядок производной)

Решение:

1. Находим решение соответствующего ЛОДУ (приравниваем левую часть уравнения к нулю)
2. Составляем
3. Применяем метод вариации произвольных постоянных:
   1. Делаем замену
   2. Получаем, что
   3. Составим условия:
      1. …
   4. Находим , из них находим , получаем решение ДУ, подставляя их в (1).

Доказательство: нет оснований не верить Колотию.

# Основы теории вероятностей и статистических методов

## 

## 25 Биномиальное распределение. Распределение Пуассона. Теорема Пуассона (с доказательством).

**Биномиальное распределение** описывает количество успехов в n независимых испытаниях, каждое с вероятностью успеха p. Если X — случайная величина, представляющая количество успехов, то X распределена биномиально: X ~ Bin(n, p). Функция вероятности для биномиального распределения:, где и .

**Распределение Пуассона** описывает количество событий, происходящих в фиксированный интервал времени или пространства, при условии, что события происходят с постоянной средней скоростью λ и независимо друг от друга. Если Y — случайная величина, представляющая количество событий, то Y распределена по Пуассону: Y ~ Poisson(λ). Функция вероятности для распределения Пуассона:

**Теорема Пуассона** утверждает, что биномиальное распределение при больших n и малых p приближается к распределению Пуассона с параметром λ = np. Доказательство:

Рассмотрим биномиальную случайную величину X ~ Bin(n, p). Требуется показать, что при n → ∞ и p → 0 таким образом, что λ = np остается постоянным, биномиальное распределение сходится к распределению Пуассона.

Функция вероятности биномиального распределения:

Используем аппроксимации для больших n:

При n → ∞ и p → 0 с λ = np:

Тогда:

Заменим p = λ / n:

Таким образом, при n → ∞ биномиальное распределение Bin(n, p) с параметрами n и p, где λ = np, сходится к распределению Пуассона с параметром λ:

Таким образом, теорема Пуассона доказана.

## 26 Математическое ожидание дискретной случайной величины. Свойства математического ожидания дискретной случайной величины. Математическое ожидание распределения Пуассона и распределения Бернулли

**Математическое ожидание** дискретной случайной величины – это сумма парных произведений всех возможных ее значений на соответствующие вероятности:

M(X) = Mx = x1p1 + x2p2 + … xnpn = xipi , где pi = 1

Очевидно, математическое ожидание случайной величины X не изменится, если таблицу значений этой случайной величины пополнить конечным числом любых чисел, считая, что вероятности этих чисел равны нулю.

Математическое ожидание M(X) случайной величины есть величина постоянная и поэтому представляет числовую характеристику случайной величины X.

***Вероятностный смысл математического ожидания***: математическое ожидание приближенно равно среднему арифметическому наблюдаемых значений случайной величины.

Прежде, чем формулировать свойства математического ожидания необходимо выяснить смысл и дать определение арифметических операций X+Y, X-Y, X\*Y и т.п., где X и Y – дискретные случайные величины.

Например, под суммой X+Y понимается случайная величина Z, значениями которой являются все допустимые суммы = + , где xi и yj – все возможные значения соответственно случайных величин X и Y; причем соответствующие вероятности равны:

= P(Z=) = P(X = )P(Y = / X = )

Если какая-нибудь комбинация + невозможна, то условно полагают ; это не отразится на математическом ожидании суммы. Аналогично определяются и остальные операции.

**Свойства математического ожидания случайной величины**

1. Математическое ожидание постоянной величины равно ей самой: M[C]=C, C – постоянная;
2. M[C•X]=C•M[X]
3. Математическое ожидание суммы случайных величин равно сумме их математических ожиданий: M[X+Y]=M[X]+M[Y]
4. Математическое ожидание произведения независимых случайных величин равно произведению их математических ожиданий: M[X•Y]=M[X]•M[Y], если X и Y независимы.

**Закон распределения Бернулли**

Биноминальные распределения

К этому распределению приводит **схема Бернулли**: пусть производится n независимых однородных испытаний, в каждом из которых событие A может произойти с вероятностью p(A) = p, а ему противоположное – с вероятностью p() = 1 - p = q. Рассмотрим теперь дискретную случайную величину E , равную числу появлений события A при n испытаниях. Возможными значениями E являются все целые числа от 0 до n , а вероятность того, что E примет значение m , определяется **формулой Бернулли:**

p(E=m)=

*Формула Бернулли удобна для вычислений лишь при сравнительно небольшом числе испытаний n. При больших значениях n пользоваться этой формулой неудобно. Если количество испытаний n достаточно велико, а вероятность p появления события A в отдельно взятом испытании весьма мала (0,05-0,1 и меньше), то вероятность того, что в данной серии испытаний событие A появится ровно m раз, можно приближенно вычислить по формуле Пуассона.*

**Распределение Пуассона**

Дискретная случайная величина называется распределенной по закону Пуассона, если её возможными значениями являются все целые неотрицательные числа (0, 1, …), а вероятность того, что случайная величина Е примет значение m, определяется формулой Пуассона:

p(E=m) = , где a = np

## 27 Ковариация случайных величин. Свойства ковариации случайных величин. Ковариация независимых случайных величин (с доказательством). Матрица ковариаций.

**Ковариа́ция** или **корреляционный момент** c o v ( X , Y ) случайных величин — в [теории вероятностей](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D0%BE%D1%8F%D1%82%D0%BD%D0%BE%D1%81%D1%82%D0%B5%D0%B9) и [математической статистике](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%81%D1%82%D0%B0%D1%82%D0%B8%D1%81%D1%82%D0%B8%D0%BA%D0%B0) мера [зависимости](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%B7%D0%B0%D0%B2%D0%B8%D1%81%D0%B8%D0%BC%D0%BE%D1%81%D1%82%D1%8C_(%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D0%BE%D1%8F%D1%82%D0%BD%D0%BE%D1%81%D1%82%D0%B5%D0%B9)) двух [случайных величин](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D0%B0%D1%8F_%D0%B2%D0%B5%D0%BB%D0%B8%D1%87%D0%B8%D0%BD%D0%B0).

В теории вероятностей и статистике ковариация является мерой совместной изменчивости двух случайных величин. Если большие значения одной переменной в основном соответствуют большим значениям другой переменной, и то же самое верно для меньших значений (то есть переменные имеют тенденцию одинаковой направленности) — ковариация положительна. При отрицательной ковариации большие значения одной переменной в основном соответствуют меньшим значениям другой и наоборот (то есть переменные имеют тенденцию противоположной направленности). Величину ковариации труднее интерпретировать, поскольку она не нормирована и, следовательно, зависит от размерности величин. Нормализованная версия ковариации — [коэффициент корреляции](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D1%80%D1%80%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D1%8F) — своей величиной показывает силу линейной зависимости.

Пусть X , Y — две случайные величины, определённые на одном и том же [вероятностном пространстве](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D1%80%D0%BE%D1%8F%D1%82%D0%BD%D0%BE%D1%81%D1%82%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%BE). Тогда их ковариация определяется следующим образом:

c o v ( X , Y ) = E [ ( X − E X ) ( Y − E Y ) ] ,

где E — [математическое ожидание](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%BE%D0%B6%D0%B8%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5) (в русскоязычной литературе также используется обозначение M ).

Предполагается, что все [математические ожидания](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%BE%D0%B6%D0%B8%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5) E в правой части данного выражения определены.

**Ковариацио́нная ма́трица** (или **ма́трица ковариа́ций**) в [теории вероятностей](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D0%BE%D1%8F%D1%82%D0%BD%D0%BE%D1%81%D1%82%D0%B5%D0%B9) — это [матрица](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), составленная из попарных [ковариаций](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B2%D0%B0%D1%80%D0%B8%D0%B0%D1%86%D0%B8%D1%8F) элементов одного или двух [случайных векторов](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D0%BC%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D0%B0%D1%8F_%D0%B2%D0%B5%D0%BB%D0%B8%D1%87%D0%B8%D0%BD%D0%B0).

Ковариационная матрица [случайного вектора](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D0%B0%D1%8F_%D0%B2%D0%B5%D0%BB%D0%B8%D1%87%D0%B8%D0%BD%D0%B0#%D0%A2%D0%B8%D0%BF%D1%8B_%D1%81%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D1%8B%D1%85_%D0%B2%D0%B5%D0%BB%D0%B8%D1%87%D0%B8%D0%BD) — квадратная симметрическая неотрицательно определенная матрица, на диагонали которой располагаются [дисперсии](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BF%D0%B5%D1%80%D1%81%D0%B8%D1%8F_%D1%81%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D0%BE%D0%B9_%D0%B2%D0%B5%D0%BB%D0%B8%D1%87%D0%B8%D0%BD%D1%8B) компонент вектора, а внедиагональные элементы — ковариации между компонентами.

Ковариационная матрица случайного вектора является многомерным аналогом дисперсии случайной величины для случайных векторов. Матрица ковариаций двух случайных векторов — многомерный аналог ковариации между двумя случайными величинами.

В случае нормально распределённого случайного вектора ковариационная матрица вместе с математическим ожиданием этого вектора полностью определяют его распределение (по аналогии с тем, что математическое ожидание и дисперсия [нормально распределённой](https://ru.wikipedia.org/wiki/%D0%9D%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%80%D0%B0%D1%81%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5) случайной величины полностью определяют её распределение)

* Пусть X : Ω → R n , Y : Ω → R m — два случайных вектора размерности n и m соответственно. Пусть также случайные величины X i , Y j , i = 1 , … , n , j = 1 , … , m имеют конечный второй [момент](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%BC%D0%B5%D0%BD%D1%82%D1%8B_%D1%81%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D0%BE%D0%B9_%D0%B2%D0%B5%D0%BB%D0%B8%D1%87%D0%B8%D0%BD%D1%8B) ([дисперсию](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BF%D0%B5%D1%80%D1%81%D0%B8%D1%8F_%D1%81%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D0%BE%D0%B9_%D0%B2%D0%B5%D0%BB%D0%B8%D1%87%D0%B8%D0%BD%D1%8B)), то есть X i , Y j ∈ L 2 . Тогда матрицей ковариации векторов X , Y называется

Σ = c o v ( X , Y ) = E [ ( X − E X ) ( Y − E Y ) ⊤ ] ,

то есть

Σ = ( σ i j ) ,

где

σ i j = c o v ( X i , Y j ) ≡ E [ ( X i − E X i ) ( Y j − E Y j ) ] , i = 1 , … , n , j = 1 , … , m ,

E — [математическое ожидание](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%BE%D0%B6%D0%B8%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5).

* Если X ≡ Y , то Σ называется матрицей ковариации вектора X и обозначается c o v ( X ) .[[1]](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B2%D0%B0%D1%80%D0%B8%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0#cite_note-Shiryaev-1) Такая матрица ковариации является обобщением [дисперсии](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BF%D0%B5%D1%80%D1%81%D0%B8%D1%8F_%D1%81%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D0%BE%D0%B9_%D0%B2%D0%B5%D0%BB%D0%B8%D1%87%D0%B8%D0%BD%D1%8B) для многомерной [случайной величины](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D0%B0%D1%8F_%D0%B2%D0%B5%D0%BB%D0%B8%D1%87%D0%B8%D0%BD%D0%B0), а её [след](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%B5%D0%B4_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D1%8B) — скалярным выражением [дисперсии](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BF%D0%B5%D1%80%D1%81%D0%B8%D1%8F_%D1%81%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D0%BE%D0%B9_%D0%B2%D0%B5%D0%BB%D0%B8%D1%87%D0%B8%D0%BD%D1%8B) многомерной [случайной величины](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D0%B0%D1%8F_%D0%B2%D0%B5%D0%BB%D0%B8%D1%87%D0%B8%D0%BD%D0%B0). В связи с этим используется также обозначение V ( X ) — дисперсия случайного вектора. [Собственные векторы](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%B1%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D1%8B%D0%B5_%D0%B2%D0%B5%D0%BA%D1%82%D0%BE%D1%80%D1%8B) и [собственные числа](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%B1%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D1%8B%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%B0) этой матрицы позволяют оценить размеры и форму облака распределения такой [случайной величины](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D0%B0%D1%8F_%D0%B2%D0%B5%D0%BB%D0%B8%D1%87%D0%B8%D0%BD%D0%B0), аппроксимировав его эллипсоидом (или эллипсом в двумерном случае).
* Сокращённая формула для вычисления матрицы ковариации:

c o v ( X ) = E [ X X ⊤ ] − E [ X ] ⋅ E [ X ⊤ ] .

* Матрица ковариации случайного вектора [неотрицательно определена](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%BE%D0%B6%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE_%D0%BE%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D1%91%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0)[[1]](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B2%D0%B0%D1%80%D0%B8%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0#cite_note-Shiryaev-1):

c o v ( X ) ≥ 0 .

* Смена масштаба:

c o v ( a ⊤ X ) = a ⊤ c o v ( X ) a , ∀ a ∈ R n .

* Если случайные векторы X и Y нескоррелированы ( c o v ( X , Y ) = 0 ), то

c o v ( X + Y ) = c o v ( X ) + c o v ( Y ) .

* Матрица ковариации [аффинного преобразования](https://ru.wikipedia.org/wiki/%D0%90%D1%84%D1%84%D0%B8%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%B5%D0%BE%D0%B1%D1%80%D0%B0%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5):

c o v ( A X + b ) = A c o v ( X ) A ⊤ ,

где A — произвольная матрица размера n × n , а b ∈ R n .

* [Перестановка](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0) аргументов:

c o v ( X , Y ) = c o v ( Y , X ) ⊤

* Матрица ковариации [аддитивна](https://ru.wikipedia.org/wiki/%D0%90%D0%B4%D0%B4%D0%B8%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D1%81%D1%82%D1%8C_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) по каждому аргументу:

c o v ( X 1 + X 2 , Y ) = c o v ( X 1 , Y ) + c o v ( X 2 , Y ) ,

c o v ( X , Y 1 + Y 2 ) = c o v ( X , Y 1 ) + c o v ( X , Y 2 ) .

* Если X и Y независимы, то

c o v ( X , Y ) = 0 .

# Дискретная математика

## 28 Совершенные дизъюнктивная и конъюнктивная нормальные формы представления булевых функций. Построение СДНФ и СКНФ по таблице истинности.

СДНФ - дизъюнкция всех минтермов.

Т. Любую ф-ию f(x1…xn) можно представить в форме СДНФ единственным образом, при этом 2 любые СДНФ задают две разные ф-ии.

Док-во: раскладываем по Шенону по всем переменным одновременно - разложение единственно - Ч.т.д.

построение СДНФ по таблице истинности

Совершенная дизъюнктивная нормальная форма (СДНФ) - это представление булевой функции в виде дизъюнкции всех возможных комбинаций литералов, при которых функция принимает значение 1. То есть, если булевая функция f(x1, x2, ..., xn) имеет таблицу истинности, то ее СДНФ можно получить следующим образом:

1. Перечисляем все строки таблицы истинности, в которых функция принимает значение 1.

2. Для каждой строки записываем дизъюнкцию литералов, соответствующих значениям переменных в данной строке.

3. Объединяем все полученные дизъюнкции с помощью знака "или".

| x1 | x2 | x3 | f | СДНФ |
| --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 1 | (-x1)(-x2)(-x3) |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | (-x1)x2x3 |
| 1 | 0 | 0 | 1 | x1(-x2)(-x3) |
| 1 | 0 | 1 | 1 | x1(-x2)x3 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | x1x2x3 |

f(x1, x2, x3) = (¬x1 ∧ ¬x2 ∧ ¬x3) ∨ (¬x1 ∧ x2 ∧ x3) ∨ (x1 ∧ ¬x2 ∧ ¬x3) ∨ (x1 ∧ ¬x2 ∧ x3) ∨ (x1 ∧ x2 ∧ x3)

Совершенная конъюнктивная нормальная форма (СКНФ) - это представление булевой функции в виде конъюнкции всех возможных комбинаций литералов, при которых функция принимает значение 0. То есть, если булевая функция f(x1, x2, ..., xn) имеет таблицу истинности, то ее СКНФ можно получить следующим образом:

1. Перечисляем все строки таблицы истинности, в которых функция принимает значение 0.

2. Для каждой строки записываем конъюнкцию литералов, соответствующих значениям переменных в данной строке.

3. Объединяем все полученные конъюнкции с помощью знака "и".

Пример: для той же таблицы истинности, что и выше, ее СКНФ будет выглядеть следующим образом:

f(x1, x2, x3) = (¬x1 ∨ ¬x2 ∨ x3) ∧ (x1 ∨ ¬x2 ∨ x3) ∧ (x1 ∨ x2 ∨ ¬x3)

## 29 Полнота систем булевых функций. Классы функций, сохраняющих 0, сохраняющих 1. Самодвойственные, монотонные и линейные функции. Формулировка теоремы о функциональной полноте.

Множество F называется *функционально полным*, если любую функцию F из , где - универсальное мн-во U, можно представить как суперпозицию функции из F.

Множество F называется *функционально полным*, если любая ф-ия из представима как формула над F.

Для доказательства ф.п. достаточно показать, что системой функций можно выразить любую другую ф.п. систему (мн-во функций).

1. {}- мн-во ф. п.  
    Док-во:  
    СДНФ Ч.Т.Д.
2. {}- ф.п.  
    Док-во:   
   G = {g1, g2}: g1=xy, g2=  
   F = {f1, f2, f3}: f1=xy, f2=xy, f3=  
   f1 = g1(x,y)  
   f2=f((x)(y)) = g2(g1(g2(x),g2(y)))  
   f3 = g2(x)

Ч.Т.Д.

*Замкнутость класса функций:*

Мн-во называется замкнутым классом функций, если любая суперпозиция данных функций и есть функция из F.  
Прим.: 0(f1), 1(f2), x(f3), x(f4) => f3(f1(x)) = f0 класс {0,1,x,x} - замкнут.

*Функции сохраняющие 0:*

Функция F называется сохраняющей 0, если F(0,0,...0)=0.  
Обозначается . ( - замкнут.)  
Пример: конъюнкция , сумма по модулю 2 .

*Функции сохраняющие 1:*

Булева функция называется сохраняющей 1, если F(1,1,...,1) = 1. Обозначается . ( - замкнут.)  
Пример: дизъюнкция , импликация   
 Функция называется *двойственной* для другой функции, когда она получается из отрицания функции и задающих ее значений.  
 f  
 Функция называется *самодвойственной* если она равна двойственной к ней.  
 Мн-во самодвойственных функций: , . Обозначается буквой S. (S - замкнут.)   
 Свойства:  
1. Противоположные наборы находятся симметрично относительно середины таблицы истинности;  
2. На противоположных наборах самодвойственная функция принимает противоположные значения.

*Монотонные функции:*  
 Функция называется монотонной, если для любой пары сравнимых наборов входных переменных x и y, (векторов x,y), так, что для всех i, значение функции для x меньше или равно значению функции для y.  
 Обозначается М. (М - замкнут.)  
 Пример: конъюнкция, дизъюнкция, константы.

*Линейные функции:*  
 Ранг терма - кол-во литералов в этом терме. Ранг формулы над термами - ранг максимального терма.  
 Функция называется линейной, если ранг ее полинома Жегалкина <=1.  
(Полином Жегалкина не содержит произведений)

Для : f(x) =   
 f(0) = = 1 => = 1  
 f(1) = = = 0 => =1  
 Получаем: 1x => линейная   
Обозначается L. (L - замкнут.)  
Пример: сумма по модулю, инверсия.

Классы:

Теорема Поста о полноте: Система функций F будет функционально полной тогда и только тогда, когда для каждого класса Поста найдется такая которая не будет ему принадлежать.   
 (То есть для нашей системы функций хотя бы одна не будет принадлежать , не будет принадлежать , S, L, M).

## 30 Основные тавтологии исчисления высказываний. Аксиомы и правила вывода исчисления высказываний.

1) Закон исключенного третьего P\lor\lnot P

2) Закон отрицания противоречия \lnot(P\land\lnot P)

3) Закон двойного отрицания \lnot\lnot P\leftrightarrow P

4) Закон тождества P\to P

5) Закон контрапозиции (P\to Q)\leftrightarrow(\lnot Q\to\lnot P)

6) Закон силлогизма (правило цепного заключения) ((P\to Q)\land(Q\to R))\to (P\to R)

7) Закон противоположности (P\leftrightarrow Q)\leftrightarrow (\lnot P\leftrightarrow\lnot Q)

8) Правило добавления антецедента («Истина из чего угодно») P\to(Q\to P)

9) Правило «из ложного что угодно» \lnot P\to(P\to Q)

10) Правило modus ponens (P\land(P\to Q))\to Q

11) Правило modus tollens ((P\to Q)\land\lnot Q)\to\lnot P

12) Правило перестановки посылок(P\to (Q\to R))\leftrightarrow (Q\to (P\to R))

13) Правило объединения (и разъединения) посылок (P\to (Q\to R))\leftrightarrow ((P\land Q)\to R)

14) Правило разбора случаев ((P\to R)\land (Q\to R))\leftrightarrow ((P\lor Q)\to R)

15) Правило приведения к абсурду ((\lnot P\to Q)\land (\lnot P\to\lnot Q))\to P,~ (\lnot P\to (Q\land\to Q))\to P

## 31 Формулы логики предикатов. Равносильность формул, включающих кванторы существования и всеобщности, знаки конъюнкции, дизъюнкции и импликации.

Для записи формул логики предикатов пользуются следующими символами:

1) *p*, *q*, *r*, … – переменные высказывания, которые принимают значения 0 или 1;

2) *x*, *y*, *z*, … – предметные переменные, которые принимают значения из некоторого множества *М*. *x*0, *y*0, *z*0, … – предметные константы, т. е. значения предметных переменных;

3) *Р*(·), *F*(·), … – одноместные предикаты,

*Р*(·,·, …,), *F*(·,·, …,) – *n*-местные предикаты,

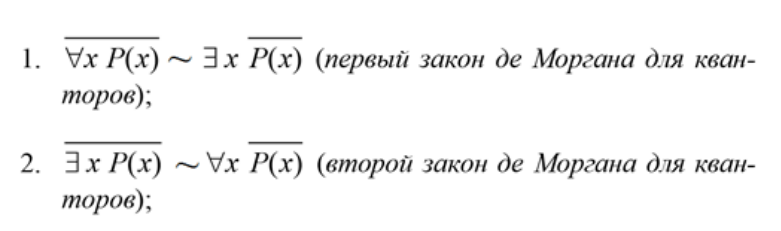
*Р*0(·), *Р*0(·,·,…,) – постоянные предикаты;

4) Символы логических операций: ;

5) Символы кванторных операций:  

6) Вспомогательные символы: скобки ( ), запятые.

Пусть P(x) , Q(x) и R(x, y) — произвольные два одноместных предиката и двухместный предикат, а S — произвольное переменное высказывание (или формула, не содержащая x). Тогда имеют место следующие равносильности



## 32 Основные понятия теории графов. Деревья, соотношение между количеством вершин и ребер (доказательство). Связность графов, мосты и точки сочленения. Неравенства для вершинной и реберной связности.

**Граф** - математический объект, состоящий из множеств V и E, обозначается как G(V, E). V - множество вершин, E - множество ребер, соединяющих их. **Ребро** - кортеж из двух вершин.

Соединенные вершины называются **соседними**. Количество ребер, выходящих из вершины называется ее **степенью**. **Путь** - последовательность смежных ребер. **Цикл** - путь, в котором начальная и конечная вершины совпадают.

**Связный граф** - путь, в котором есть путь от любой вершины до любой.

**Дерево** - связный граф без циклов.

Соотношение: Если в дереве n вершин, то в нем n-1 ребер.

Док-во: По индукции. База индукции: рассм дерево с 1 вершиной, без ребер (кол-во ребер = кол-ву вершин -1). Переход: предположим, что верно для всех деревьев с кол-вом вершин <= n, где n > 1. Докажем, что верно и для n+1: Добавим новую вершину к дереву: чтобы оно осталось деревом, нужно добавить ровно одно ребро. Таким образом, при добавлении вершины увеличивается количество вершин и ребер, и условие остается верным.

**Мост** - ребро, удаление которого увеличивает число компонент связности графа.

**Точка сочленения** - вершина, удаление которой (вместе с ребрами) увеличивает число компонент связности графа.

**Вершинная связность** – мин. кол-во вершин, которые нужно удалить, чтобы граф стал несвязным.

**Реберная связность** - аналогично

Для графа с n вершинами вершинная и реберная связность не превышают (n-1). Т.к. макс кол-во ребер, которые можно удалить, оставив граф связным, равно “кол-ву вершин -1”.

Для любого связного графа вершинная и реберная связность не могут быть больше мин степени вершины в графе. Док-во: можно просто удалить все ребра, исходящие из вершины с минимальной степенью, и граф станет несвязным.

## 33 Паросочетание в двудольном графе. Нахождение совершенных паросочетаний.

K-раскраска графа - разбиение множества вершин на k подмножеств, таких что внутри подмножества любая пара вершин несмежна.

Минимальное число k, для которого существует k-раскраска графа - хроматическое число данного графа.

Граф двудольный, если его хроматическое число равняется 2.

Паросочетание - множество попарно несмежных ребер этого графа.

Совершенное паросочетание - паросочетание, которое является минимальным реберным покрытием графа.

В двудольном графе совершенное паросочетание существует тогда и только тогда, когда какое бы множество вершин мы не взяли, выполняется условие: ,

где - множество вершин таких, что

**Нахождение совершенного паросочетания:**

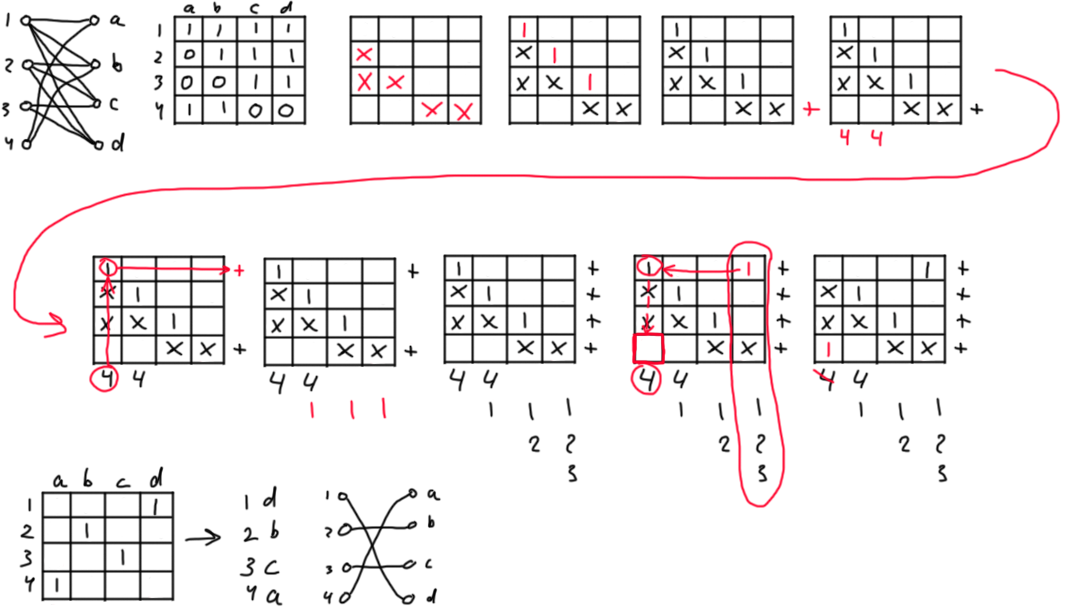
Строим матрицу смежности двудольного графа, где по одной оси вершины первой доли, на другой - вершины другой доли (вершины внутри своих долей несмежны между собой по определению, поэтому такие варианты не учитываем при построении матрицы).

Далее строим матрицу аналогичной смежности. На новой матрице блокируем ячейки, которые равны 0 в первой матрице.

Теперь заполняем новообразованную матрицу единицами по правилу "в каждом столбце и каждой строке может находиться только одна единица" случайным образом пока это получается (например, идя по строкам снизу вверх, в каждой строке заполняем единицей самую левую доступную для заполнения ячейку (которая изначально не заблокирована и в столбце которой уже не стоит единица)).

На следующем этапе находим строку без единиц и помечаем ее. В помеченной строке под всеми столбцами, доступными для заполнения, пишем номер данной строки, помечая данные столбцы. Теперь, для каждого помеченного столбца ищем строку с единицей, помечаем ту строку и повторяем вышеуказанные действия. Действуем так до тех пор, пока все строки и столбцы не окажутся помеченными.

В какой-то момент образуется помеченный столбец, в котором нет единицы. Ставим единицу в любую доступную строку этого столбца. Единицу, которая уже находилась на этой строке, необходимо переместить на другую строку. Список строк для перемещения указан под столбцом с этой единицей (он образовался вследствие этапа с пометками как раз). После перемещения номер выбранной для перемещения строки вычеркивается из списка. Если единица была перемещена на строку, на которой уже была другая единица, для этой другой единицы проводим тот же процесс перемещения.



Источник: [Совершенное паросочетание в двудольном графе](https://youtu.be/QxUYvqI8wBg?si=--N-ftQfbPx6HN9n)

## 34 Понятие планарности графа. Соотношение между количеством вершин, рёбер и граней в плоском графе. Теорема Пантрягина-Куратовского. Род графа.

**Плоским** графом называется граф, изображенный на плоскости так, что никакие два его ребра геометрически не пересекаются нигде, кроме инцидентной им обоим вершины.

Граф называется ***планарным***, если он изоморфен плоскому графу.

**Формула Эйлера:** Для всякого связного плоского графа верно равенство: n – m + f = 2, где n – число вершин m — число ребер, a f— число граней графа.

**Теорема Понтрягина–Куратовского.** Для того чтобы граф G был планарным, необходимо и достаточно, чтобы он не содержал ни одного подграфа, гомеоморфного графам K5 или K3,3.

**Род графа** — это минимальное целое число n, такое, что граф может быть нарисован без пересечения самого себя на сфере с n маркерами (т. е. ориентированной поверхностью рода n)

К примеру плоский граф имеет род 0, потому что его можно нарисовать на сфере без самопересечения

## 35 Раскраски графов. Хроматическое число, теоремы о пяти красках (с доказательством) и о четырёх красках (формулировка).

Раскрасить граф значит поставить каждой вершине в соответствие некоторый цвет таким образом, чтобы смежные вершины были разных цветов.

Минимальное число K, для которого существует K-раскраска графа G, называется хроматическим числом графа λ(G).

Для связных планарных графов справедлива формула:

n + r = m + 2, где n - число вершин, m - число рёбер, r - число граней

Доказательство:

Пусть дан планарный связный граф G. Построим его остовное дерево Gk. У любого дерева будет ровно одна грань, а m = n - 1, поэтому формула n + 1 = n - 1 + 2 справедлива

Далее будем по очереди добавлять рёбра (удалённые во время формирования остовного дерева).

Т.к. граф планарен, то добавление циклического ребра будет увеличивать количество граней на 1.

Тогда после восстановления всех удалённых рёбер графа получим:

n + k + 1 = n + k - 1 + 2

k + 1 = r, n + k - 1 = m

n + r = m + 2

ч.т.д

Следствие 1:

Для любого планарного графа с n ≥ 3 вершинами справедливо неравенство:

m ≤ 3n - 6

Доказательство:

Пусть идя по i-й грани мы проходим L(i) рёбер.

Ребро в планарном графе либо лежит внутри какой-то грани и при переходе через это ребро мы остаёмся в этой грани, либо оно разделяет две грани. Тогда отсюда следует, что если мы просуммируем L(i) всех граней, то получим 2m.

При этом любая грань ограничена как минимум 3 рёбрами. Так же просуммируем L(i) для всех граней, только возьмём l(i) = 3 для всех i в качестве нижней границы.

Отсюда: 3r <= 2m. Выразим r через формулу Эйлера.

3(m + 2 - n) <= 2m, m <= 3n - 6. Следствие доказано.

Следствие 2.

В любом плоском графе хотя бы одна вершина имеет степень ≤ 5.

Доказательство:

Допустим, что это не так. Пусть каждая вершина имеет степень ≥ 6.

Тогда, по лемме о рукопожатиях, сумма степеней всех вершин = 2m.

С другой стороны, опять же, возьмём в качестве нижней границы 6 и просуммируем для всех вершин минимальную степень 6.

Отсюда:

6n ≤ 2m ⇒ m ≥ 3n, что противоречит первому следствию.

Теорема.

Всякий планарный граф можно раскрасить в 5 цветов

Доказательство (мат. индукция)

Теорема очевидно справедлива для плоских графов с числом вершин <= 5. Пусть она верна для графов с n-1 вершинами и докажем её для произвольного графа G с n вершинами.

Действительно, в силу следствия 2, в G имеется вершина x, смежная с 0, 1, 2, 3, 4 или 5 вершинами. Подграф, получаемый удалением x, можно раскрасить в 5 цветов. Раскрасим его вершины пятью цветами α, β, γ, δ, ε. Вершину x легко окрасить всегда, кроме случаев, когда она смежна с пятью различно окрашенными вершинами.

В этом случае a, b, c, d, e - пять смежных с x вершин (в порядке обхода вокруг x (это важно!!)). И пусть α, β, γ, δ, ε их цвета. Обозначим через подграф, порождённый множеством вершин цвета α или β и т.д.

1. Если a и c принадлежат различным компонентам связности графа , то в компоненте, содержащей a перекрасим вершины цвета α в цвет γ, и наоборот. Новая окраска совместима с графом G, а вершину x, которая окружена теперь вершинами с окраской γ, β, γ, δ, ε, можно окрасить в цвет α

2. Если a и c принадлежит одной и той же компоненте графа G\_α,γ, то вершины b и d нельзя соединить в графе . Переставляя цвета β и δ в той компоненте графа , которая содержит вершину b, мы получим возможность окрасить x в цвет β.

Теорема.

Всякий планарный граф можно раскрасить в 4 цвета.

## 

## 36 Применение производящих функций для решения линейных рекуррентных соотношений с постоянными коэффициентами.

## 

## 37 Специальные комбинаторные числа: Стирлинга 1-го и 2-го рода, Бэлла и Каталана.

Обозначение. (t)n=t(t-1)…(t-n+1).

Числа Стирлинга определяются следующим образом. Положим

(t)0=t0=s(0,0)=S(0,0)=1

(t)n=t(t-1)…(t-n+1)=, n>0 (1)

tn=, n>0. (2)

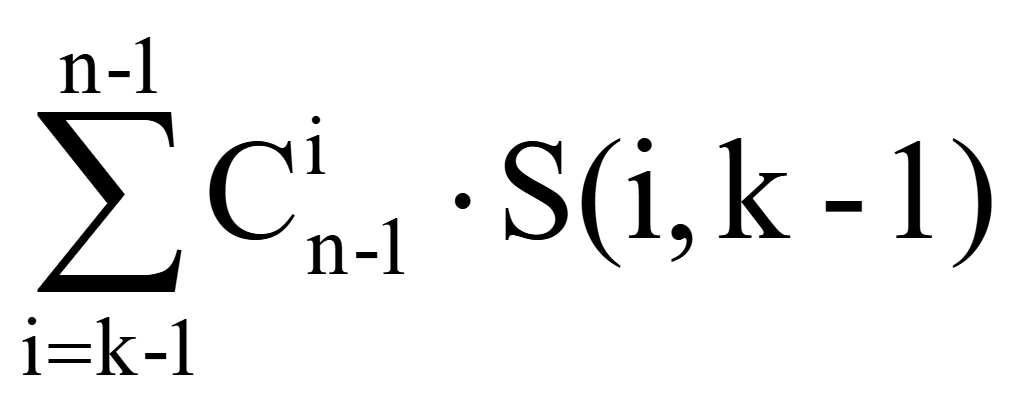
Тогда s(n,k) и S(n,k) называются числами Стирлинга соответственно первого и второго рода. Заметим, что числа обоих рядов отличаются от нуля только для k=1,2,…,n, n>0 и что (t)n является обычной производящей функцией для s(n,k), тогда как tn является новым типом производящей функции для входящей в уравнение (\*) функции fk(t), равной (t)k.

С числами Стирлинга второго рода можно связать разбиения конечных множеств, а именно:

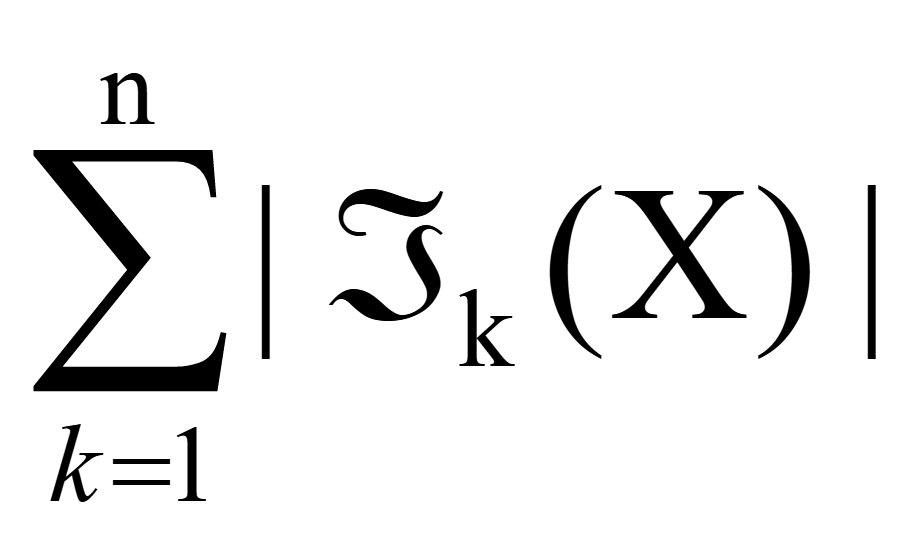
Пусть Х={1,2,…,n}, рассмотрим всевозможные разбиения Х на k блоков. Множество таких разбиений будем обозначать Пk(X), пусть u(n,k)=|Пk(X)|, тогда

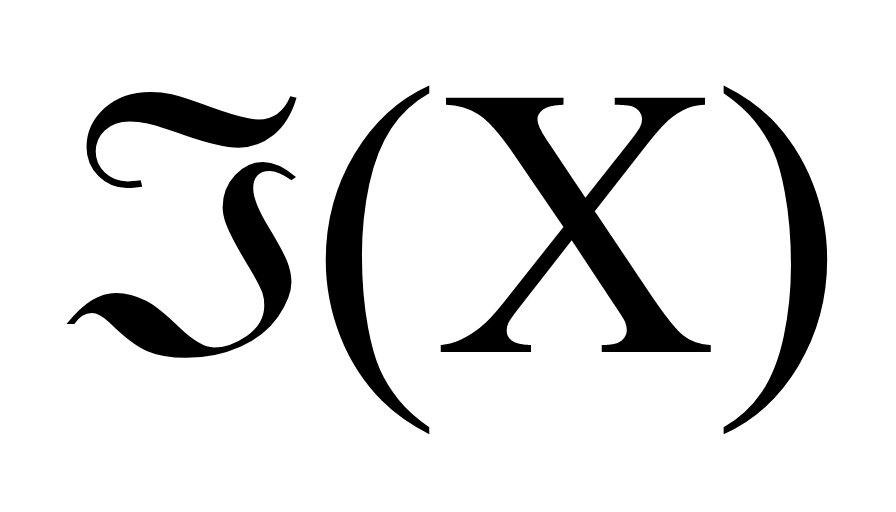
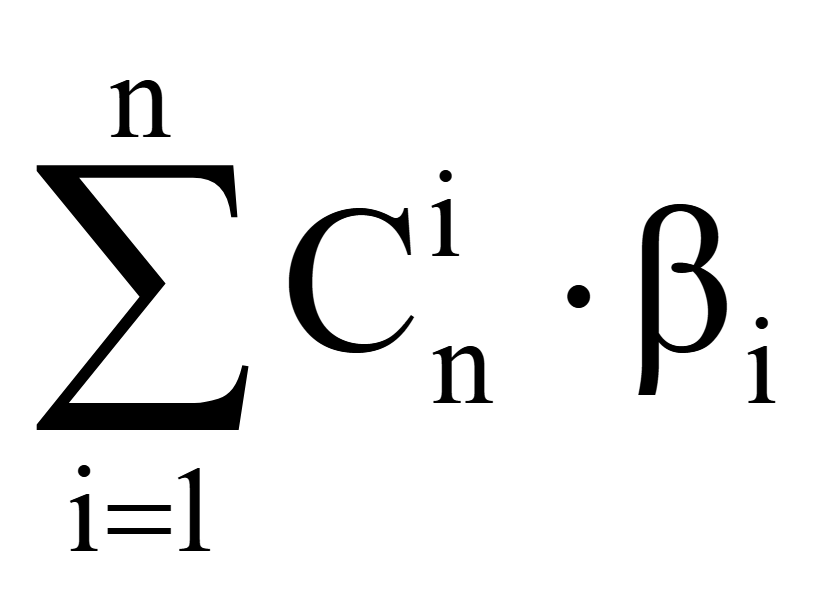
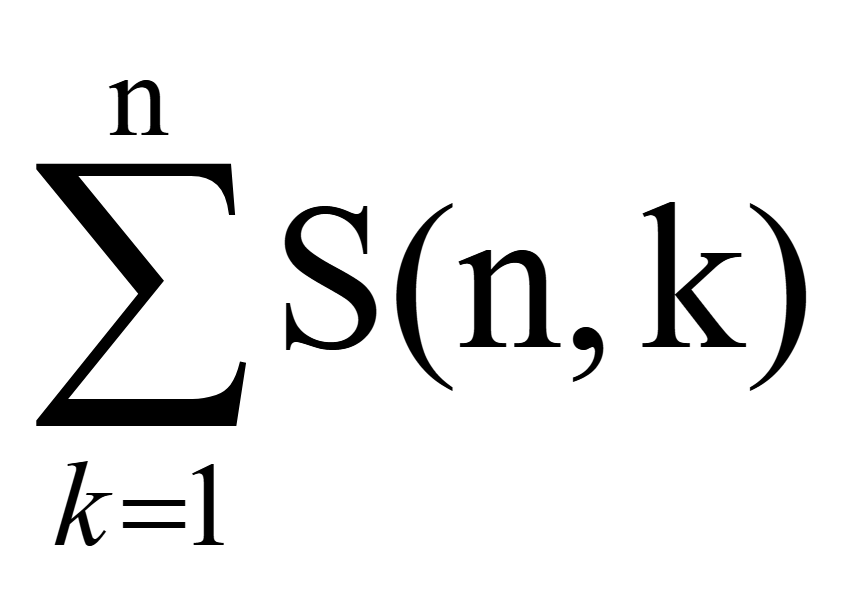
u(0,0)=1 u(n,k)=u(n-1,k-1)+ku(n-1,k).

Числа Стирлинга 2-го рода удовлетворяют тождеству:



S(n,k) = , k≥2

*Число Белла* βn определяется как число разбиений n-элементного множества, т. е.

βn=||= , другими словами, βn= 

Справедливо следующее тождество: βn=

Пусть |X|=n, |Y|=k, то число всех функций f:XY и f(X)=Y, равно Sn,k=k!S(n,k).

ГОСЫ ГЛАВА 2

# Конструирование алгоритмов и структур данных

## 1 Понятие сортировки массива. Оценка алгоритмов сортировки. Сортировка Хоара.

Пусть на некотором мн-ве X задано отношение <=, обладающее св-вами:

1.a,b, a<=b||b<=a

2.a, a<=a

3.a,b,c, a<=b && b<=c a<=c

4.a,b, a<=b && b<=a a=b

Т.е на множестве X задано отношение порядка

пусть a1,...,an, тогда i1,..,in - такая перестановка чисел 1,...,n , что последовательность ai1,..,ain такова, что вып-ся св-во ai1<=ai2<=...<=ain

Эта посл наз-ся упорядоченной. Методы, позволяющие построить упорядоченную последовательность наз-ся **методами сортировки** . Большинство задач сортировки рассматриваются на некотором абстракции последовательности и представляется в виде

| Pa+a1 | Key | Pa+a2 | Key |
| --- | --- | --- | --- |

В каждом эл-те посл выделяется или назначается отдельное поле Key, имеющее уникальное значение. Именно по этому полю вып-ся сортировка

Различают внешние и внутренние алгоритмы сортировки.Если сортируемый набор данных полностью помещается в оперативную память, то можно применять внутренние методы сортировки, в противном случае - внешние.

**Оценивание алгоритмов сортировки**

При разработке алгоритмов необходимо учитывать его эффективность с точки зрения вычислительных ресурсов системы.

Рассматривают 2 основных ресурса:

1)Время работы процессора

2)Основная память, отводимая под данные алгоритма

Время выполнения цикла:

t=c\*n , c- время выполнения 1 итерации (не зависит от n)

Для определения эффективности работы алгоритма достаточно указать, как зависит время работы алгоритма от n.

Рассматривают асимптотическую эффективность алгоритма т.е. как зависит время работы алгоритма от количества входных данных, растущих до бесконечности.

Пусть время работы алгоритма выражено f(n), будем рассматривать скорость роста функции f(n) с помощью упрощенной функции T(n)

f(n)=an^2+bn+c T(n)=n^2 O(n^2)- грубая оценка

при упрощенной функции можно пренебречь значениями a,b,c,n

Существуют 3 меры сложности алгоритма:

1)В худшем случае

2)В среднем 50/50

3)Средне-взвешенная

**Сортировка Хоара(Быстрая сортировка)**

Данная сортировка за счет рекурсии позволяет сократить время обработки до O ( n\*)

В сортируемом наборе выделяется эл-т с номером n , такой ,что все эл-ты , расположенные левее него имеют значения ключей меньше , чем ключ f , а значения правее - больше, т.е. эл с номером f находится на своем месте в отсортированной последовательности . Такой элемент называется опорным элементом. Следовательно в алгоритме нужно поменять элементы местами так, чтобы они заняли свою половину относительно опорного эл-та. Если опорный элемент отсутствует, то за несколько перестановок его можно получить , а затем рекурсивно применить алгоритм к каждой из половин.

Код:

int a[20] ,n;

void hoarsort(int\*,iny,int);

void main(){

int i ;

cin>>n;

for(i=0;i<n;i++)

cin>>a[i]

hoarsort(a,o,n-1);

for(i-0;i<n;i++)

cout<<a[i];}

void hoarsort(int\*a,int first,int second){

int i=first;

j=second;

int tmp;

int f=(first+second)/2];

do

{

while (a[i]<f) i++;

while(a[j]>f) j–;

if(i<=j)

{

if(i<j){

tmp=a[i]; a[i]=a[j]; a[j]=tmp;

}

i++;j –;

}

} while(i<=j);

if(i<second)

hoarsort(a,i,second);

if (first<j)

hoarsort(a,first,j);}

## 2. Статическая и динамическая память в C++. Программное построение бинарных деревьев поиска.

**Статическая и динамическая память в C++**

В языке программирования C++ память может быть выделена двумя основными способами: статически и динамически.

**Статическая память:**

- Статическая память выделяется на стадии компиляции и существует в течение всего времени выполнения программы.

- Переменные, которые определены вне функций или в функциях с ключевым словом `static`, размещаются в статической памяти.ьцу

- Пример:

static int staticVar = 10; // Статическая переменная

**Динамическая память:**

- Динамическая память выделяется и освобождается в ходе выполнения программы с использованием операторов `new` и `delete`.

- Динамическая память позволяет управлять объемом памяти, используемой программой, во время её выполнения.

- Пример:

int\* dynamicVar = new int; // Выделение динамической памяти

\*dynamicVar = 20; // Использование динамической памяти

delete dynamicVar; // Освобождение динамической памяти

**Программное построение бинарных деревьев поиска (BST)**

Бинарное дерево поиска (Binary Search Tree, BST) — это бинарное дерево, в котором для каждого узла выполняется следующее условие:

- Значения всех узлов в левом поддереве меньше значения текущего узла.

- Значения всех узлов в правом поддереве больше значения текущего узла.

**Определение узла BST:**

struct Node {

int data;

Node\* left;

Node\* right;

Node(int value) : data(value), left(nullptr), right(nullptr) {}

};

**Вставка элемента в BST:**

Node\* insert(Node\* root, int value) {

if (root == nullptr) {

return new Node(value);

}

if (value < root->data) {

root->left = insert(root->left, value);

} else {

root->right = insert(root->right, value);

}

return root;

}

**Поиск элемента в BST:**

bool search(Node\* root, int value) {

if (root == nullptr) {

return false;

}

if (root->data == value) {

return true;

} else if (value < root->data) {

return search(root->left, value);

} else {

return search(root->right, value);

}

}

**Удаление элемента из BST:**

Node\* findMin(Node\* root) {

while (root->left != nullptr) {

root = root->left;

}

return root;

}

Node\* deleteNode(Node\* root, int value) {

if (root == nullptr) {

return root;

}

if (value < root->data) {

root->left = deleteNode(root->left, value);

} else if (value > root->data) {

root->right = deleteNode(root->right, value);

} else {

if (root->left == nullptr) {

Node\* temp = root->right;

delete root;

return temp;

} else if (root->right == nullptr) {

Node\* temp = root->left;

delete root;

return temp;

}

Node\* temp = findMin(root->right);

root->data = temp->data;

root->right = deleteNode(root->right, temp->data);

}

return root;

}

## 3. Контейнеры языка С++: векторы, множества, мультимножества, очереди. Связное распределение памяти. Однонаправленные списки. Операции вставки и удаления элемента.

Контейнеры языка С++ векторы, множества, мультимножества, очереди.

В языке C++ существует множество контейнеров, которые предоставляют различные способы организации и управления данными. Вот некоторые из них:

*Векторы (Vectors):*

Векторы представляют собой динамические массивы, которые могут изменять свой размер во время выполнения программы. Они обеспечивают быструю вставку и доступ к элементам по индексу, но могут быть медленными при удалении или вставке элементов в середину контейнера.

Пример использования вектора:

#include <vector>

... // какой-то код

std::vector<int> myVector;

myVector.push\_back(10); // Вставка элемента в конец вектора

myVector.pop\_back(); // Удаление последнего элемента

*Множества (Sets):*

Множества представляют собой контейнеры, которые хранят уникальные элементы в отсортированном порядке. Они предоставляют эффективный поиск элементов, но не поддерживают доступ к элементам по индексу.

Пример использования множества:

#include <set>

... // какой-то код

std::set<int> mySet;

mySet.insert(10); // Вставка элемента в множество

mySet.erase(10); // Удаление элемента из множества

*Мультимножества (Multisets):*

Мультимножества похожи на множества, но позволяют хранить неуникальные элементы. Они также поддерживают отсортированное расположение элементов.

Пример использования мультимножества:

#include <multiset>

... // какой-то код

std::multiset<int> myMultiset;

myMultiset.insert(10); // Вставка элемента в мультимножество

myMultiset.erase(10); // Удаление элемента из мультимножества

*Очереди (Queues):*

Очереди представляют собой контейнеры, работающие по принципу "первым пришёл – первым вышел" (First-In-First-Out, FIFO). Элементы добавляются в конец очереди, а удаление происходит из начала очереди.

Пример использования очереди:

#include <queue>

... // какой-то код

std::queue<int> myQueue;

myQueue.push(10); // Добавление элемента в очередь

myQueue.pop(); // Удаление первого элемента из очереди

Связное распределение памяти.

Связное распределение памяти (Linked Allocation) – это метод организации памяти, в котором объекты хранятся в виде независимых участков памяти, называемых узлами (nodes), а связь между узлами устанавливается с помощью указателей. Каждый узел содержит данные и указатель на следующий узел. Это позволяет эффективно вставлять и удалять элементы из контейнера, но требует дополнительной памяти для хранения указателей.

Однонаправленные списки.

Однонаправленные списки – это структура данных, состоящая из узлов, где каждый узел содержит данные и указатель на следующий узел. Последний узел списка указывает на nullptr, что обозначает конец списка. Однонаправленные списки позволяют эффективно вставлять и удалять элементы в начало, середину и конец списка.

Операции вставки и удаления элемента.

*Операции вставки и удаления элементов в однонаправленных списках:*

Вставка элемента в однонаправленный список:

· Создайте новый узел и запишите в него данные.

· Установите указатель нового узла на следующий узел после вставляемого узла.

· Установите указатель предыдущего узла на новый узел.

· Обновите указатель предыдущего узла на новый узел.

· Если вставка происходит в начало списка, обновите указатель на голову списка.

Удаление элемента из однонаправленного списка:

· Найдите узел, который нужно удалить.

· Обновите указатель предыдущего узла, чтобы он указывал на следующий узел после удаляемого узла.

· Удалите узел.

*Операции вставки и удаления элементов являются основными операциями, выполняемыми над контейнерами и структурами данных. В зависимости от типа контейнера или структуры данных, методы вставки и удаления могут различаться. Ниже приведены общие принципы операций вставки и удаления элементов:*

Вставка элемента:

1. Определение места вставки: Сначала необходимо определить место, куда требуется вставить элемент. Это может быть начало контейнера, конец, середина или определенное положение, определяемое, например, ключом или индексом.

2. Создание нового элемента: Затем создается новый элемент, содержащий значение, которое нужно вставить.

Размещение элемента: В зависимости от контейнера или структуры данных, новый элемент1. может быть размещен в определенной позиции путем перемещения существующих элементов или изменения указателей или ссылок на новый элемент.

Удаление элемента:

1. Определение места удаления: Сначала необходимо определить место, откуда требуется удалить элемент. Это может быть начало контейнера, конец, середина или определенное положение, определяемое, например, ключом или индексом.

2. Удаление элемента: Сам элемент удаляется из контейнера или структуры данных. В зависимости от типа контейнера, это может включать перемещение или перестройку оставшихся элементов, чтобы заполнить пустое место.

Важно помнить, что операции вставки и удаления могут иметь различную сложность и требования к производительности в зависимости от типа контейнера или структуры данных. Некоторые структуры данных, такие как массивы, могут иметь ограничения на вставку и удаление элементов, особенно в середине или начале, так как это может потребовать сдвига или перемещения большого количества элементов. Другие структуры данных, такие как связные списки или деревья, обеспечивают более эффективные операции вставки и удаления, но могут требовать дополнительной памяти для хранения связей или указателей.

При выборе контейнера или структуры данных для определенной задачи важно учитывать требования по скорости, эффективности использования памяти и типичные операции, выполняемые над данными.

# Интерпретируемые языки программирования

## 4. Типы данных в Python. Списки, кортежи. Основные методы. Лямбда-функции в Python. Генераторы. Итераторы. Фреймворк Django. Модели. Миграции.

Типы данных в Python:

Строковые данные (String):

Строковый тип данных используется для хранения текстовой информации. Строки в Python могут быть созданы с помощью одинарных, двойных или тройных кавычек.

Основные методы:

- `capitalize()`: Преобразует первую букву строки в верхний регистр.

- `upper()`: Преобразует все буквы строки в верхний регистр.

- `lower()`: Преобразует все буквы строки в нижний регистр.

- `strip()`: Удаляет пробельные символы в начале и конце строки.

- `split()`: Разбивает строку на список подстрок по указанному разделителю.

- `join()`: Объединяет элементы списка в одну строку, используя текущую строку как разделитель.

Числовые данные (Numeric):

Числовые типы данных используются для хранения числовой информации. Основные числовые типы в Python:

- Целые числа (int): Представляют целые числа без десятичной части. Пример: `x = 5`.

- Числа с плавающей точкой (float): Представляют числа с десятичной частью. Пример: `y = 3.14`.

- Комплексные числа (complex): Представляются в виде `a + bj`, где `a` - это реальная часть, а `b` - мнимая часть. Пример: `z = 2 + 3j`.

Логические данные (Boolean):

Логические типы данных могут принимать только два значения: `True` (истина) или `False` (ложь). Они часто используются для условных проверок и логических операций.

Списки (Lists):

Список в Python - это упорядоченная коллекция элементов, которая может содержать объекты любого типа. Они создаются с использованием квадратных скобок `[]`. Основные методы:

- `append()`: Добавляет элемент в конец списка.

- `extend()`: Расширяет список, добавляя элементы из другого списка.

- `insert()`: Вставляет элемент в указанную позицию.

- `remove()`: Удаляет первое вхождение элемента с указанным значением.

- `pop()`: Удаляет элемент по индексу (и возвращает его).

- `index()`: Возвращает индекс первого вхождения элемента с указанным значением.

- `count()`: Возвращает количество вхождений указанного элемента.

- `sort()`: Сортирует список по возрастанию.

- `reverse()`: Разворачивает список в обратном порядке.

Кортежи (Tuples):

Кортеж в Python - это неизменяемая упорядоченная коллекция элементов. Они создаются с использованием круглых скобок `()`. Основные методы:

- `count()`: Возвращает количество вхождений указанного элемента.

- `index()`: Возвращает индекс первого вхождения элемента с указанным значением.

Лямбда-функции в Python:

Лямбда-функции - это способ определения анонимных функций в Python. Они создаются с помощью ключевого слова `lambda`. Пример:

add = lambda x, y: x + y

print(add(3, 5)) # Выведет: 8

Генераторы и итераторы:

Генераторы и итераторы - это способы эффективной работы с последовательностями данных в Python, позволяющие обрабатывать их по мере необходимости, а не загружать все сразу в память. Генераторы создаются с использованием функций с ключевым словом `yield`, а итераторы могут быть созданы с использованием протокола итерации.

Фреймворк Django:

Django - это высокоуровневый веб-фреймворк на языке Python, который позволяет быстро создавать веб-приложения. Он предоставляет множество инструментов для работы с базами данных, шаблонами, URL-адресами и т. д.

Модели и миграции в Django:

- Модели (Models): В Django модели представляют собой объектно-ориентированные классы, которые отображаются на таблицы в базе данных. Они определяют структуру данных и логику доступа к данным.

- Миграции (Migrations): Миграции в Django - это автоматически создаваемые скрипты, которые изменяют схему базы данных в соответствии с изменениями в моделях. Они позволяют разработчикам обновлять базу данных без необходимости вручную создавать SQL-скрипты.

# Инструменты проектирования ИС

## 5. Опишите, как Вы понимаете отношение между классами (обмен сообщениями). Покажите, как такое отношение обозначается на диаграмме классов. Опишите основные элементы диаграммы последовательности, покажите, как на ней отмечается обмен сообщениями. Приведите пример простейшей диаграммы последовательности.

Отношение между классами в объектно-ориентированном программировании описывает связи и взаимодействие между классами в системе. Оно может быть реализовано через обмен сообщениями между объектами классов. Обмен сообщениями происходит, когда один объект вызывает метод другого объекта для выполнения определенной операции.

На диаграмме классов отношение между классами, связанными обменом сообщений, обычно отображается с использованием ассоциаций или зависимостей. На диаграмме последовательности обмен сообщениями между объектами классов показывается последовательность вызова методов с указанием временных интервалов и параметров.

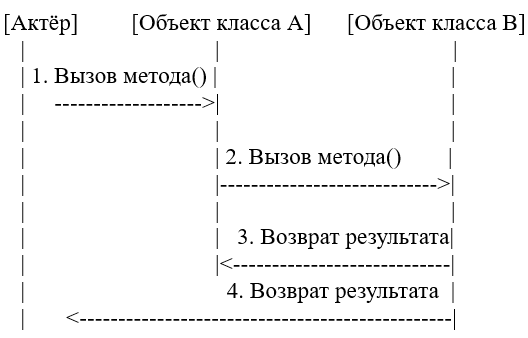
Основные элементы диаграммы последовательности:

Актеры: Представляют внешние сущности, взаимодействующие с системой. Обычно изображаются в виде прямоугольника с именем.

Объекты классов: Представляют экземпляры классов, которые обмениваются сообщениями. Обычно изображаются в виде прямоугольника с именем и стереотипом класса.

Сообщения: Показывают вызовы методов и передачу данных между объектами. Обычно изображаются стрелками, направленными вниз, с указанием имени метода и параметров.

Пример диаграммы:



## 6. Этапы и процессы жизненного цикла проекта. Каскадная модель жизненного цикла. V модель жизненного цикла. Модель ЖЦ ИС на основе прототипирования, Инкрементная и итерационная модели ЖЦ ИС, Спиральная модель жизненного цикла, Гибкие методологии проектирования. Когда какие модели целесообразно использовать.

**Жизненный цикл** традиционно разделяют на следующие основные этапы:

1. Анализ требований,

2. Проектирование,

3. Кодирование (программирование),

4. Тестирование и отладка,

5. Эксплуатация и сопровождение.

**Каскадная модель**: линейная, последовательная модель, каждый этап начинается после завершения предыдущего.

Этапы: анализ требований, проектирование, кодирование, тестирование, эксплуатация и сопровождение.

**V-модель**: последовательная модель с тестированием на каждом этапе разработки.

Этапы: анализ требований, системное проектирование, архитектурное проектирование, разработка модулей, модульное тестирование, интеграционное тестирование, системное тестирование, приемочное тестирование.

**Модель прототипирования**: создание прототипа для получения обратной связи и корректировки требований.

Этапы: планирование, быстрый анализ, создание базы данных и интерфейса, разработка функций, итерационное прототипирование.

**Инкрементная модель**: итеративный подход, каждая итерация добавляет функциональность.

Этапы: анализ, проектирование, разработка, тестирование. Каждый инкремент представляет собой работающий продукт.

**Итерационная модель**: создание продукта через последовательные итерации, каждая из которых улучшает предыдущую версию.

Используется при четко определенных конечных требованиях, для больших проектов.

**Спиральная модель**: итеративная с акцентом на анализ рисков.

Этапы: планирование, анализ рисков, конструирование, оценка результата. Подходит для сложных и критичных проектов.

**Agile модель**: гибкая методология с регулярными итерациями и адаптацией к изменениям.

Подходит для проектов с постоянно меняющимися требованиями. Использует ежедневные встречи (Scrum) и регулярные собрания (Sprint).

Когда использовать модели:

Каскадная: для проектов с четко определенными требованиями.

V-модель: для проектов, где тестирование на каждом этапе критично.

Прототипирование: для проектов, требующих частой обратной связи от пользователей.

Инкрементная: для проектов, где нужно поэтапное добавление функциональности.

Итерационная: для больших проектов с эволюционирующими требованиями.

Спиральная: для сложных, критически важных проектов.

Agile: для динамичных проектов с изменяющимися требованиями.

## 7. Диаграммы UML, типы диаграмм, статические и динамические, диаграммы классов, объектов, компонентов и развертывания. Диаграмма деятельности, диаграмма прототипов, диаграмма последовательности и аспекты ее применения.

**Диаграмма** в языке моделирования [UML](https://ru.wikipedia.org/wiki/UML) — наглядное представление некоей совокупности элементов модели системы в виде [графа](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), на котором дуги (отношения) связывают вершины (сущности). В своём графическом виде различные виды диаграмм UML ([диаграммы классов](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%BE%D0%B2), [компонентов](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0_%D0%BA%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%82%D0%BE%D0%B2), [объектов](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0_%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BE%D0%B2) и др.) применяются для визуализации разных аспектов устройства или поведения моделируемой системы

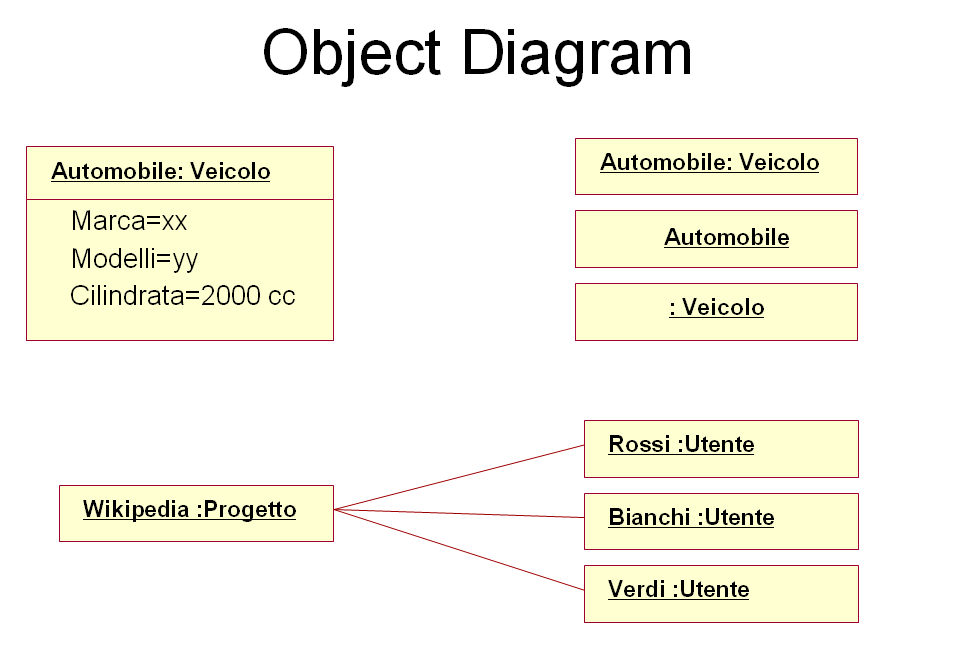
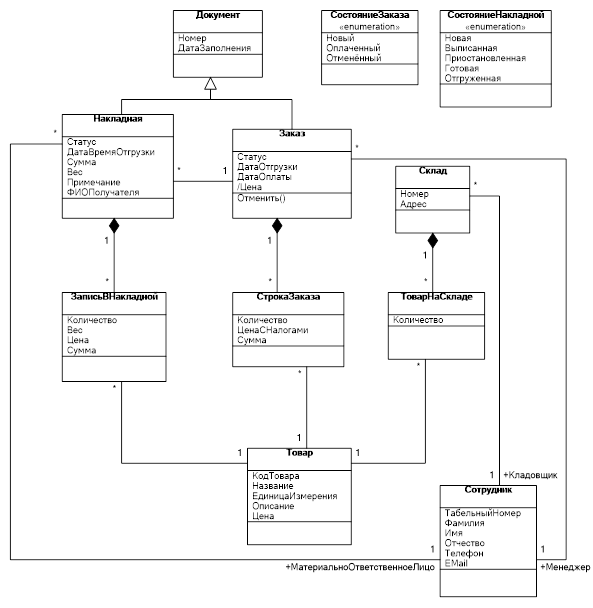
**Статические диаграммы**

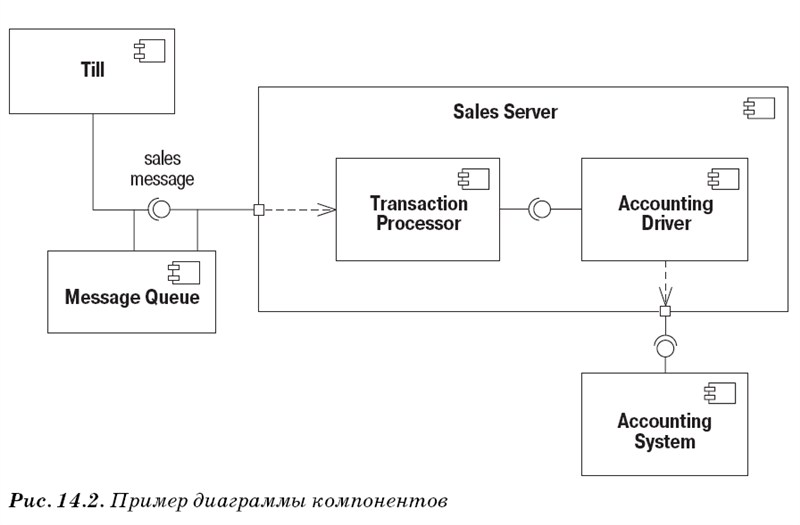
Статические диаграммы представляют либо постоянно присутствующие в системе сущности и связи между ними. К этому типу относятся диаграммы классов, объектов, компонентов и диаграммы развертывания.  
 Диаграммы классов (classdiagrams) показывают классы или типы сущностей системы, характеристики классов (поля и операции) и возможные связи между ними.  
 Диаграммы объектов (objectdiagrams) показывают часть объектов системы и связи между ними в некотором конкретном состоянии или суммарно, за некоторый интервал времени.  
 Диаграммы компонентов (componentdiagrams) представляют компоненты в нескольких смыслах — атомарные составляющие системы с точки зрения ее сборки, конфигурационного управления и развертывания.  
 Диаграммы развертывания (deploymentdiagrams) показывают декомпозицию системы на физические устройства различных видов — серверы, рабочие станции, терминалы, принтеры, маршрутизаторы и пр.

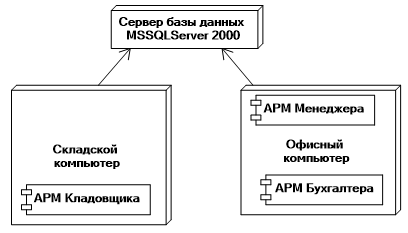
**Динамические диаграммы**

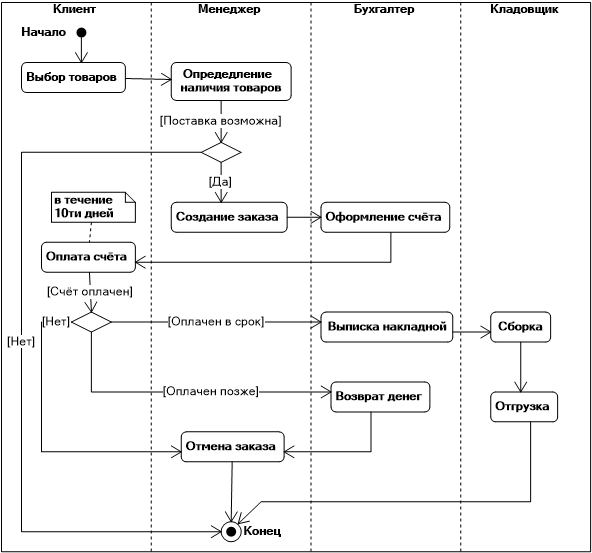
Динамические диаграммы описывают происходящие в системе процессы. К ним относятся диаграммы деятельности, сценариев, диаграммы взаимодействия и диаграммы состояний.  
 Диаграммы деятельности (activitydiagrams) иллюстрируют набор процессов-деятельностей и потоки данных между ними, а также возможные их синхронизации друг с другом.  
 Диаграммы сценариев (или диаграммы последовательности, sequencediagrams) показывают возможные сценарии обмена сообщениями или вызовами во времени между различными компонентами системы.  
 Диаграммы взаимодействия (collaborationdiagrams) показывают ту же информацию, что и диаграммы сценариев, но привязывают обмен сообщениями/вызовами не к времени, а к связям между компонентами.  
 Диаграммы состояний (statechartdiagrams) показывают возможные состояния отдельных компонентов или системы в целом, переходы между ними в ответ на какие-либо события и выполняемые при этом действия.

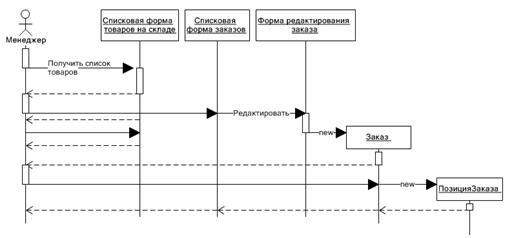
Диаграмма классов











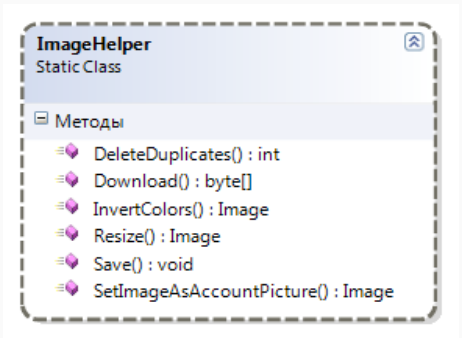
# Паттерны программирования

## 8. Перечислите принципы Solid. Приведите практические ТЕХНОЛОГИЧЕСКИЕ примеры необходимости таких принципов и способы их реализации на любом языке.

* The **S**ingle Responsibility Principle (Принцип единственной ответственности)
* The **O**pen Closed Principle (Принцип открытости/закрытости)
* The **L**iskov Substitution Principle (Принцип замещения Лисков)
* The **I**nterface Segregation Principle (Принцип разделения интерфейса)
* The **D**ependency Inversion Principle (Принцип инверсии зависимости

**1.Принцип единственной ответственности**

Класс Image Helper и его методы: InvertColor(изменить цвета на изображении), DeleteDublicates(удалить из файловой системы все дублирующиеся изображения и вернуть количество удаленных), Download(загрузка битового массива с изображением с помощью HTTP запроса), Resize(изменить размер изображения), Save - (Cохранить изображение в файловой системе), SetImageAsAccountPicture(Запрос к базе данных для сохранения ссылки на это изображение для пользователя )

**

*Класс должен быть ответственен лишь за что-то одно. Если класс отвечает за решение нескольких задач, его подсистемы, реализующие решение этих задач, оказываются связанными друг с другом. Изменения в одной такой подсистеме ведут к изменениям в другой.*

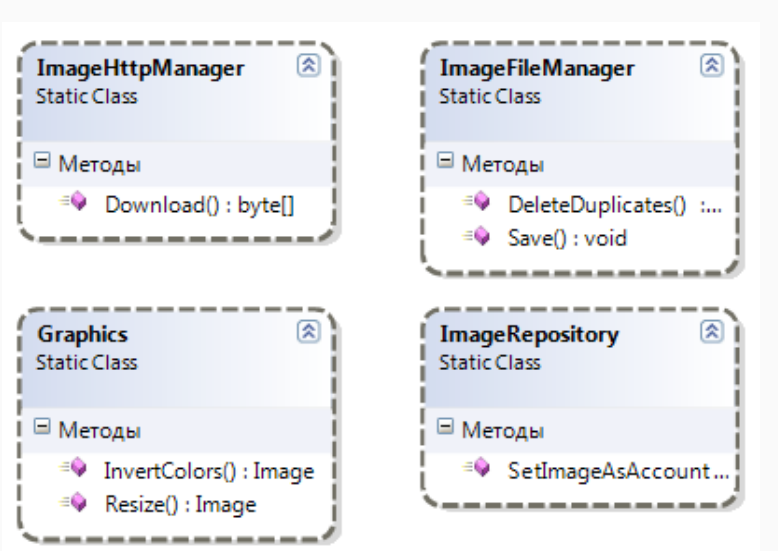
**Решение: Делим на классы отдельно**

**ImageHTTPManager:** Download

**ImageFileManager:** DeleteDuplicates, Save

**Graphics**: InvertColor, Resize

**ImageRepository**:SetImageAsAccountPicture

****

**2.Принцип открытости/закрытости**

*Программные сущности (классы, модули, функции и т.д.) должны быть открыты для расширения, но закрыты для изменения.*

Есть класс отвечающий за отрисовку фигур:

public class Shape { }

public class Square : Shape { }

public class Circle : Shape { }

public class Canvas {

public void DrawAllShapes( IEnumerable shapes ) {

foreach (var shape in shapes)

{

if (shape is Square) { //рисуем квадрат }

if (shape is Circle) { //рисуем круг }

}

}

Вопрос: а если еще добавить хотим фигуру а как

**Решение**

public interface IShape { void Draw(); }

public class Circle : Shape, IShape { public void Draw() { } }

public class Square : Shape, IShape { public void Draw() { } }

public class Canvas {

public void DrawAllShapes(IEnumerable shapes)

{

foreach (var shape in shapes) { shape.Draw(); }

}

}

**3.Принцип разделения интерфейса**

*Клиенты не должны зависеть от методов, которые они не используют. Создавайте узкоспециализированные интерфейсы, предназначенные для конкретного клиента.*

Плохая реализация(реализую метод который не нужно):

public interface IGeometry { int GetSquare(); int GetVolume(); }

public class Rectangle : IGeometry

{

public int GetSquare() { return height \* width; }

public int GetVolume() { throw new Exception("Операция не поддерживается!"); }

}

public class Parallelepiped : IGeometry {

public int GetSquare() { throw new Exception("Операция не поддерживается!"); }

public int GetVolume() { return height \* width \* depth; }

}

**Решение: делим все на блоки по функциональности**

public interface IFigure { int GetSquare(); }

public interface IBody { int GetVolume(); }

public class Rectangle : IFigure {

public int GetSquare() { return height \* width; }

}

public class Parallelepiped : IBody

{

public int GetVolume() { return height \* width \* depth; }

}

**4.Принцип замещения Лисков**

*Необходимо, чтобы подклассы могли бы служить заменой для своих суперклассов.*

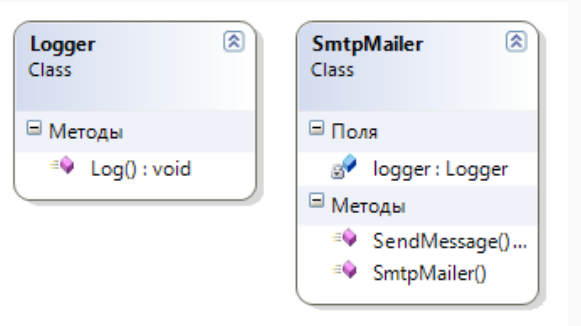
*Цель этого принципа заключаются в том, чтобы классы-наследники могли бы использоваться вместо родительских классов, от которых они образованы, не нарушая работу программы.*

Пример: **public static int DoubleValue( object a ) { return ((int)a) \* 2; }**

**5.Принцип инверсии зависимости**

*Модули верхнего уровня не должны зависеть от модулей нижнего уровня. Оба должны зависеть от абстракции. Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций*

**Пример что было:**

****

public class Logger { // сохранить лог в файле public void Log(string logText) { } }

public class SmtpMailer {

private readonly Logger logger;

public SmtpMailer() { logger = new Logger(); }

SendMessage(string message) { logger.Log(string.Format("Отправлено '{0}'", message)); } }

public class DatabaseLogger { // сохранить лог в базе public void Log(string logText) { }

}

**Решение:**public interface ILogger { void Log(string logText); }

public class Logger : ILogger { public void Log(string logText) { } }

public class SmtpMailer {

private readonly ILogger logger;

public SmtpMailer(ILogger logger) { this.logger = logger; }

public void SendMessage(string message) { logger.Log(string.Format("Отправлено '{0}'", message)); }

}

## 9. Опишите, как Вы понимаете разницу между паттернами и фреймворками. Опишите типы паттернов проектирования, приведите примеры нескольких паттернов каждого из типов, приведите тривиальный примеры.

Различие между паттернами и фреймворками состоит в следующем. Фреймворк можно рассматривать как реализацию системы паттернов проектирования (поведенческих паттернов). В то время как фреймворк — это исполняемая программа, а паттерн — это знание и опыт как решать конкретную задачу.

Существует несколько типов паттернов проектирования:

1. Порождающие. Помогают оптимизировать создание объектов и управлять их работой. Примеры: «Фабрика», «Прототип», «Строитель», «Одиночка», «Ленивая инициализация».

2. Структурные. Отвечают за то, как объекты структурированы в коде. Примеры: «Декоратор», «Компоновщик», «Мост», «Фасад», «Заместитель».

3. Поведенческие. Описывают, как объекты ведут себя и взаимодействуют с другими. Примеры: «Итератор», «Наблюдатель», «Хранитель», «Цепочка ответственности», «Посредник».

Пример: Реализация паттерна «Одиночка»

**public** **class** Singleton {

**private** **static** **final** Singleton INSTANCE = **new** Singleton();

**private** Singleton() {

}

**public** **static** Singleton getInstance() {

**return** INSTANCE;

}

}

## 10. Опишите архитектурный паттерн MVC в ООП, в чем его отличие от MVP. Опишите, как происходит разделение обязанностей между классами в рамках этого паттерна. Опишите понятие стереотипов в UML и покажите, как они используются для обозначения MVC паттерна. Приведите пример диаграммы последовательностей для реализации ЛЮБОЙ операции CRUD в рамках MVC паттерна. Опишите две известные Вам реализации MVC, в чем разница, что общее и в каких случаях рекомендовано использовать тот или иной способ.

*MVC (сокращение от Model—View—Controller)* — это архитектурный паттерн, который делит модули на три группы:

* модель (model),
* представление (view),
* контроллер (controller).

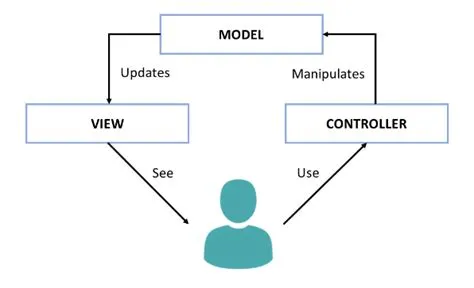
Модель содержит данные приложения, за которыми приходит пользователь. Например, список своих заказов в интернет-магазине.

Представление показывает эти данные в понятном для пользователя виде. Например, на свёрстанной странице сайта или в приложении на телефоне.

Контроллеры принимают пользовательские команды и преобразуют данные по этим командам. Например, если пользователь нажимает кнопку «Удалить заказ», то контроллер отмечает этот заказ в модели удалённым.

В архитектуре MVC пользователь взаимодействует только с представлением — чаще всего это UI.

Пользователь подаёт команды программе. Контроллер получает эти команды и преобразует данные в модели. Модель обновляется и уведомляет представление о том, что нужно перерисовать интерфейс, чтобы отобразить изменения в данных.



В классическом MVC события пользователя вызывают контроллер. Иногда события обрабатываются сперва представлением, которое затем вызывает контроллер. Оба подхода возможны и используются.

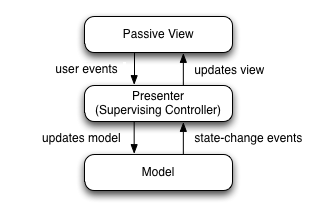
В обязанности Представления входит отображение данных полученных от Модели. Однако, представление не может напрямую влиять на модель. Можно говорить, что представление обладает доступом «только на чтение» к данным.

Представление обладает следующими признаками:

* В представлении реализуется отображение данных, которые получаются от модели любым способом;
* В некоторых случаях, представление может иметь код, который реализует некоторую бизнес-логику.

В MVP (сокращение от Model-View-Presenter) место контроллера занимает презентер.

Главное отличие от MVC в том, как расположены компоненты и, соответственно, как передаются данные. Если в MVC данные передавались по кругу, то в MVP компоненты располагаются по линии. На концах находятся модель и представление, а между ними — презентер.



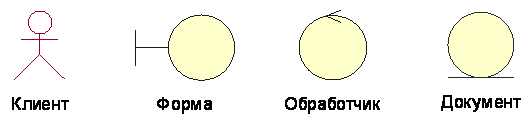
Презентер забирает на себя всю логику обработки данных, обновления представления и обработки пользовательских команд.

Представление в этом случае пассивно: оно не делает ничего, кроме отображения данных так, как ему скажет презентер. Если в MVC представление могло брать форматирование вывода на себя, то в MVP за это тоже будет отвечать презентер.

Плюс такого подхода в том, что не возникает вопросов, какой код к чему относится. Минус — в том, что презентер быстро становится большим и сложным. Приходится разбивать его на модули поменьше, вероятно, добавлять дополнительные «слои».

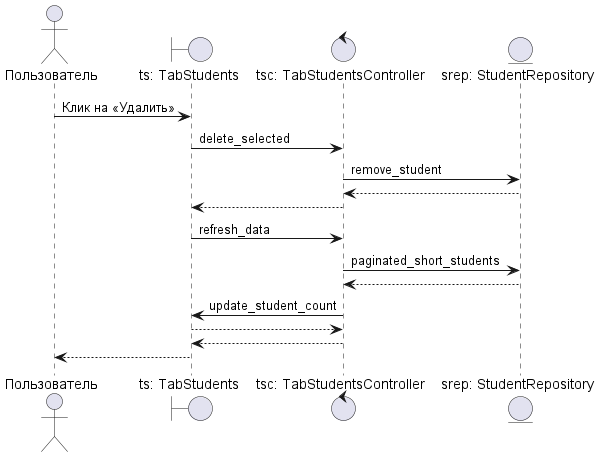
При создании диаграмм классов часто пользуются понятием «стереотип». В дальнейшем речь пойдет о стереотипах классов. Стереотип класса – это элемент расширения словаря UML, который обозначает отличительные особенности в использовании класса. Стереотип имеет название, которое задается в виде текстовой строки. При изображении класса на диаграмме стереотип показывается в верхней части класса в двойных угловых скобках. Есть четыре стандартных стереотипа классов, для которых предусмотрены специальные графические изображения.

Стереотип используется для обозначения классов-сущностей (классов данных), стреотип описывает пограничные классы, которые являются посредниками между ПС и внешними по отношению к ней сущностями – актерами, обозначаемыми стереотипом <>. Наконец, стереотип описывает классы и объекты, которые управляют взаимодействиями. Применение стереотипов позволяет, в частности, изменить вид диаграмм классов.



Стереотипы для классов

Пример диаграммы последовательности



# Организация вычислительных систем

## 11. Системы адресации в ЭВМ и структуры машинных команд. Регистры, их назначение. Система команд архитектуры Intel. Язык ассемблера архитектуры Intel: описание данных, арифметические команды, команды управления.

**Системы адресации в ЭВМ:**

1. Прямая адресация: Операнды команды содержат прямые адреса ячеек памяти, где расположены данные.

2. Непосредственная адресация: Операнды команды содержат сами данные, а не их адреса.

3. Косвенная адресация: Операнды содержат адреса ячеек памяти, в которых хранятся адреса данных.

4. Регистровая адресация: Операнды указывают на регистры процессора, содержащие данные.

5. Базовая/смещенная адресация: Адреса вычисляются путем суммирования базового адреса и смещения.

6. Индексированная адресация: Адреса вычисляются путем суммирования базового адреса и значения индексного регистра.

**Структуры машинных команд:**

1. Опкод (операционный код): Определяет выполняемую операцию.

2. Операнды: Аргументы, над которыми выполняется операция.

3. Префиксы команд: Дополнительные инструкции, определяющие специальные режимы работы процессора или изменяющие поведение команды.

4. Режим адресации: Определяет, как интерпретировать операнды команды.

**Регистры и их назначение:**

1. Регистры общего назначения: Используются для временного хранения данных и адресов. Примеры: EAX, EBX, ECX, EDX в архитектуре Intel.

2. Регистры сегментов: Хранят адреса базовых адресов сегментов в памяти. Примеры: CS (Code Segment), DS (Data Segment), SS (Stack Segment) в архитектуре Intel.

3. Регистры индексов и указателей: Используются для адресации данных в памяти. Примеры: ESI (Source Index), EDI (Destination Index), ESP (Stack Pointer), EBP (Base Pointer) в архитектуре Intel.

4. Флаговые регистры: Хранят состояние процессора и результаты операций. Пример: регистр EFLAGS в архитектуре Intel.

**Система команд архитектуры Intel:**

Система команд архитектуры Intel x86 включает широкий набор инструкций, включая:

1. Описание данных: Команды для работы с данными в регистрах и памяти (например, MOV для перемещения данных, PUSH и POP для работы со стеком).

2. Арифметические команды: Команды для выполнения арифметических операций (например, ADD, SUB, MUL, DIV).

3. Логические и побитовые операции: Команды для выполнения логических и побитовых операций (например, AND, OR, XOR, SHIFT).

4. Команды перехода и условные команды: Команды для управления потоком выполнения программы (например, JMP для безусловного перехода, CMP для сравнения, JZ, JNZ для условного перехода).

5. Команды управления: Команды для управления процессором и выполнением программы (например, CALL для вызова процедур, RET для возврата из процедур, INT для вызова прерываний).

**Язык ассемблера архитектуры Intel** используется для написания программ, использующих инструкции процессора архитектуры Intel. Код на языке ассемблера состоит из мнемоник (описания инструкций), операндов (данных или адресов) и директив (инструкций ассемблера). Примеры арифметических команд: ADD (сложение), SUB (вычитание), MUL (умножение), DIV (деление). Примеры команд управления: JMP (безусловный переход), CALL (вызов процедуры), RET (возврат из процедуры).

# Современные концепции программирования

## 12. Основные концепции ООП, перечислить, дать краткую характеристику. Сравнительная характеристика различных объектно-ориентированных языков. Абстрактный базовый класс и интерфейс, примеры практического применения.

**Основные концепции ООП:**

1. **Инкапсуляция:**

- Описание объекта включает его данные и методы, которые могут манипулировать этими данными. Инкапсуляция позволяет скрыть детали реализации объекта от внешнего мира и предоставить доступ к ним только через установленные интерфейсы.

1. **Наследование**:

- Механизм, позволяющий одному классу (подклассу) наследовать свойства и методы другого класса (родительского класса). Это позволяет создавать иерархии классов, обобщать поведение и повторно использовать код.

1. **Полиморфизм**:

- Способность объектов различных классов отвечать на одни и те же сообщения с различными реакциями. Полиморфизм позволяет использовать общий интерфейс для работы с разными объектами.

**Сравнительная характеристика различных объектно-ориентированных языков:**

1. **Java**:

- Строго типизированный, объектно-ориентированный язык. Поддерживает все основные концепции ООП. Имеет механизм интерфейсов для реализации полиморфизма.

1. **Python**:

- Динамически типизированный язык с поддержкой ООП. Поддерживает инкапсуляцию и наследование, но полиморфизм реализуется несколько иначе, чем в языках со строгой типизацией.

1. **C++**:

- Мультипарадигменный язык программирования с поддержкой как процедурного, так и объектно-ориентированного программирования. Позволяет реализовывать высокоуровневые концепции ООП, а также работать с памятью напрямую.

**Абстрактный базовый класс и интерфейс:**

1. **Абстрактный базовый класс:**

- Это класс, который содержит абстрактные методы (методы без реализации) и может содержать реализованные методы. Он предоставляет общий интерфейс для всех его подклассов. Абстрактные классы могут иметь конструкторы, а интерфейсы - нет.

1. **Интерфейс:**

- Это контракт, который определяет, какие методы должен реализовать класс. Интерфейс не содержит реализации методов, только их сигнатуры. Классы могут реализовывать несколько интерфейсов.

**Примеры практического применения:**

1. **Абстрактный базовый класс:**

- Представим систему для управления транспортными средствами. У нас может быть абстрактный базовый класс "Транспортное средство", который содержит общие свойства и методы для всех видов транспорта (например, методы "двигаться()" или "остановиться()"), а также абстрактные методы, такие как "заправить()" или "ремонт()".

from abc import ABC, abstractmethod

class Transport(ABC):

def \_\_init\_\_(self, brand, model, color):

self.brand = brand

self.model = model

self.color = color

@abstractmethod

def move(self):

pass

@abstractmethod

def stop(self):

pass

@abstractmethod

def refuel(self):

pass

@abstractmethod

def repair(self):

pass

class Car(Transport):

def \_\_init\_\_(self, brand, model, color):

super().\_\_init\_\_(brand, model, color)

self.fuel\_level = 100

def move(self):

print(f"The {self.color} {self.brand} {self.model} is moving.")

def stop(self):

print(f"The {self.color} {self.brand} {self.model} stopped.")

def refuel(self):

self.fuel\_level = 100

print(f"The {self.color} {self.brand} {self.model} has been refueled.")

def repair(self):

print(f"The {self.color} {self.brand} {self.model} has been repaired.")

class Bike(Transport):

def \_\_init\_\_(self, brand, model, color):

super().\_\_init\_\_(brand, model, color)

def move(self):

print(f"The {self.color} {self.brand} {self.model} is cycling.")

def stop(self):

print(f"The {self.color} {self.brand} {self.model} stopped.")

def refuel(self):

print("Bikes don't need refueling.")

def repair(self):

print(f"The {self.color} {self.brand} {self.model} has been repaired.")

# Пример использования классов

car = Car("Toyota", "Camry", "blue")

car.move()

car.stop()

car.refuel()

car.repair()

bike = Bike("Giant", "Talon", "red")

bike.move()

bike.stop()

bike.refuel()

bike.repair()

1. **Интерфейс**:

- Допустим, у нас есть классы "Круг" и "Прямоугольник", представляющие геометрические фигуры. Мы можем создать интерфейс "Фигура", который определяет методы, такие как "площадь()" и "периметр()", и затем оба класса могут реализовать этот интерфейс, обеспечивая общий способ вычисления площади и периметра.

from abc import ABC, abstractmethod

from math import pi

class Shape(ABC):

@abstractmethod

def area(self):

pass

@abstractmethod

def perimeter(self):

pass

class Circle(Shape):

def \_\_init\_\_(self, radius):

self.radius = radius

def area(self):

return pi \* self.radius \*\* 2

def perimeter(self):

return 2 \* pi \* self.radius

class Rectangle(Shape):

def \_\_init\_\_(self, width, height):

self.width = width

self.height = height

def area(self):

return self.width \* self.height

def perimeter(self):

return 2 \* (self.width + self.height)

# Пример использования классов

circle = Circle(5)

print("Площадь круга:", circle.area())

print("Периметр круга:", circle.perimeter())

rectangle = Rectangle(4, 6)

print("Площадь прямоугольника:", rectangle.area())

print("Периметр прямоугольника:", rectangle.perimeter())

# Программирование в компьютерных сетях

## 13. Поясните понятие DOM. Способы программной навигации по HTML странице посредством DOM-дерева. Опишите механизм работы сессий и их предназначение. Для чего нужен хеш, cookie, сессии? Авторизация с помощью сессий. Методы защиты сессий.

Document Object Model (DOM) — это кроссплатформенный и независимый от языка интерфейс, который рассматривает документ HTML или XML как древовидную структуру, в которой каждый узел является объектом, представляющим часть документа. DOM представляет собой документ с логическим деревом. Каждая ветвь дерева заканчивается узлом, и каждый узел содержит объекты. Методы DOM обеспечивают программный доступ к дереву; С их помощью можно изменить структуру, стиль или содержание документа. К узлам могут быть прикреплены обработчики событий. Как только событие активируется, выполняются обработчики событий.

В рамках Javascript навигация по DOM дереву выполняется с помощью следующих методов:

1. document.getElementById(id) - Этот метод используется для получения ссылки на элемент по его уникальному идентификатору (id).
2. document.getElementsByClassName(className) - Этот метод возвращает коллекцию элементов, которые имеют определенный класс.
3. document.getElementsByTagName(tagName) - Этот метод возвращает коллекцию элементов с определенным тегом (например, "div", "p", "a").
4. document.querySelector(selector) - Этот метод возвращает первый элемент, соответствующий заданному CSS-селектору.
5. document.querySelectorAll(selector) - Этот метод возвращает все элементы, соответствующие заданному CSS-селектору.
6. parentNode - Свойство parentNode позволяет получить родительский элемент для текущего элемента в DOM-дереве.
7. childNodes - Свойство childNodes возвращает коллекцию дочерних элементов текущего элемента.
8. nextSibling и previousSibling - Свойства nextSibling и previousSibling позволяют получить следующий и предыдущий узлы на том же уровне в дереве DOM.

Сессии представляют собой механизм управления состоянием между сервером и клиентом в веб-приложениях. Они позволяют серверу идентифицировать и взаимодействовать с конкретным пользователем в течение периода времени, называемого "сессией". Вот как обычно работает процесс:

1. Создание сессии: Когда пользователь впервые посещает веб-сайт, сервер создает уникальную сессию для этого пользователя.
2. Хранение данных: Сервер сохраняет информацию о состоянии сессии, такую как переменные, привязанные к этой сессии. Эти данные могут включать в себя информацию о пользователе, его предпочтениях, корзине покупок и т. д.
3. Идентификация: Для идентификации сессии обычно используется уникальный идентификатор, который передается между сервером и клиентом.
4. Хеш и Cookie: Хеш и cookie играют важную роль в механизме сессий. Хеш-идентификатор сессии хранится на сервере и/или передается в качестве cookie на сторону клиента. Этот хеш используется для связывания клиентской сессии с серверной. Cookie представляют собой механизм хранения данных на стороне клиента, который браузер отправляет обратно серверу с каждым запросом, позволяя серверу идентифицировать сессию.
5. Авторизация с помощью сессий: После того как пользователь прошел аутентификацию (ввел логин и пароль), сервер обычно создает сессию для этого пользователя и передает ему уникальный идентификатор сессии в виде cookie. При последующих запросах клиент отправляет этот cookie, и сервер использует его для определения пользователя и его прав.

Защита сессий: Для обеспечения безопасности сессий используются различные методы:

1. Шифрование: Часто сессионные данные шифруются для предотвращения их перехвата и взлома.
2. SSL/TLS: Использование протокола HTTPS обеспечивает защищенную передачу данных между клиентом и сервером.
3. Уникальные идентификаторы: Использование длинных и случайных идентификаторов сессий уменьшает вероятность подбора сессионной куки.
4. Ограничение времени сессии: Сессии могут иметь ограниченное время жизни, после истечения которого они автоматически удаляются, чтобы предотвратить злоупотребление.
5. Блокировка сессии: После успешной аутентификации сессия может быть связана с IP-адресом или другими параметрами, чтобы предотвратить перехват и использование сессии злоумышленником с другого устройства.

# Оценка сложности алгоритмов

## 14. Понятия временной и емкостной сложности алгоритма. Функции сложности, сложность данных. Минимальная, максимальная и средняя оценки сложности. Методы анализа временной сложности алгоритмов, содержащих повторяющиеся действия (циклы типа for, циклы типа while, repeat).

*Временная сложность* – функция от размера входа, отражающая количество операций, выполненных в ходе работы алгоритма при решении задачи данного размера.

*Ёмкостная сложность* – функция от размера входа, отражающая количество ячеек памяти, к которым было обращение при решении задачи данного размера.

Минимальная, максимальная, средняя сложность - добавляем в начало определений эти слова.

Если есть циклы, временная сложность тела цикла умножается на количество раз, которое цикл выполнится.

## 15. Анализ сложности рекурсивных алгоритмов. Линейная рекурсия, нелинейная рекурсия, рекуррентные уравнения для функций сложности.

**Линейная рекурсия**

Линейная рекурсия в алгоритмах происходит, когда каждый вызов функции ведет к единственному новому рекурсивному вызову. Простым примером является вычисление факториала числа n, когда каждый вызов функции приводит к одному новому вызову этой же функции с аргументом n-1. В этом случае сложность алгоритма определяется количеством вызовов, что прямо пропорционально входному значению n, отсюда и следует линейная сложность O(n).

Рекуррентное уравнение для вычисления факториала будет иметь вид: T(n) = T(n-1) + O(1). Здесь O(1) означает, что затраты времени на каждый не рекурсивный шаг алгоритма постоянны.

**Нелинейная рекурсия**

Нелинейная рекурсия возникает, когда функция вызывает сама себя более одного раза. Примером такого алгоритма может служить быстрая сортировка или вычисление чисел Фибоначчи с использованием простого рекурсивного алгоритма. Сложность таких алгоритмов часто оказывается выше, чем линейная.

Для быстрой сортировки рекуррентное уравнение будет иметь вид: T(n) = 2T(n/2) + O(n). Здесь 2T(n/2) отражает два рекурсивных вызова функции для каждого из половинного массива, а O(n) обозначает время, затрачиваемое на разделение массива на две части. Решение этого уравнения, с использованием метода дерева рекурсии или мастер-теоремы, дает O(n log n), что говорит о логарифмической сложности алгоритма.

**Рекуррентные уравнения**

Рекуррентные уравнения - это уравнения, которые определяют каждый последующий член последовательности через предыдущие. Они используются для описания сложности рекурсивных алгоритмов, поскольку позволяют выразить затраты времени на обработку входных данных через затраты времени на обработку меньших входных данных.

Важно помнить, что рекуррентные уравнения могут иметь различные формы в зависимости от специфики алгоритма, и разные методы могут быть применены для их решения. Однако, во многих случаях, они позволяют получить оценку сложности алгоритма, которая помогает при выборе наиболее эффективного решения.

## 16. Понятие сложности задач. Классы сложности задач: класс задач полиномиальной сложности, экспоненциальной сложности и класс NP. Гипотеза P≠NP. Класс NP- полных задач: определение, теорема о сводимости (с доказательством).

1. **В начале 1960-х годов, в связи с началом широкого использования вычислительной техники для решения практических задач, возник вопрос о границах практической применимости данного алгоритма решения задачи в смысле ограничений на ее размерность. Какие задачи могут быть решены на ЭВМ за реальное время?  
   Ответ на этот вопрос был дан в работах Кобмена (Alan Cobham, 1964), и Эдмнодса (Jack Edmonds, 1965), где были введены сложностные классы задач.  
   *1) Класс P (задачи с полиномиальной сложностью)*Задача называется полиномиальной, т.е. относится к классу P, если существует константа k и алгоритм, решающий задачу с (n)=O(), где n - длина входа алгоритма в битах n = |D| [6].  
   Задачи класса P – это интуитивно, задачи, решаемые за реальное время.**
   * **Отметим следующие преимущества алгоритмов из этого класса:  
     для большинства задач из класса P константа k меньше 6;**
   * **класс P инвариантен по модели вычислений (для широкого класса моделей);**
   * **класс P обладает свойством естественной замкнутости (сумма или произведение полиномов есть полином).**
2. **Таким образом, задачи класса P есть уточнение определения «практически разрешимой» задачи.  
   *2) Класс NP (полиномиально проверяемые задачи)*Представим себе, что некоторый алгоритм получает решение некоторой задачи – соответствует ли полученный ответ поставленной задаче, и насколько быстро мы можем проверить его правильность?  
   Рассмотрим, например задачу о сумме:  
   Дано N чисел – А = (a1,…an) и число V.  
   Задача: Найти вектор (массив) X=(x1,…,xn), xi {0,1}, такой, что  = V.  
   Содержательно: может ли быть представлено число V в виде суммы каких-либо элементов массива А.  
   Если какой-то алгоритм выдает результат – массив X, то проверка правильности этого результата может быть выполнена с полиномиальной сложно-стью: проверка  aixi = V требует не более Q (N) операций.  
   Формально: Dє, |D|=n поставим в соответствие сертификат Sє , такой что |S|=O () и алгоритм  =  (D,S), такой, что он выдает «1», если решение правильно, и «0», если решение неверно. Тогда задача принадлежит классу NP, если F (  )=O ( ).  
   Содержательно задача относится к классу NP, если ее решение некоторым алгоритмом может быть быстро (полиномиально) проверено.**
3. **Проблема P = NP  
   После введения в теорию алгоритмов понятий сложностных классов Эдмондсом (Edmonds, 1965) была поставлена основная проблема теории сложности – P = NP ? Словесная формулировка проблемы имеет вид: можно ли все задачи, решение которых проверяется с полиномиальной сложностью, решить за полиномиальное время ?  
   Очевидно, что любая задача, принадлежащая классу P, принадлежит и классу NP, т.к. она может быть полиномиально проверена – задача проверки решения может состоять просто в повторном решении задачи.  
   На сегодня отсутствуют теоретические доказательства как совпадения этих классов (P=NP), так и их несовпадения. Предположение состоит в том, что класс P является собственным подмножеством класса NP, т.е. NP \ P не пусто – рис 6.1**

**Будем говорить, что имеет место полиномиальная сводимость языка к языку , если существует функция , которая удовлетворяет двум условиям:**

**1. Существует ДМТ-программа, вычисляющая функцию с временной сложностью, ограниченной полиномом.**

**2. Для любого , .**

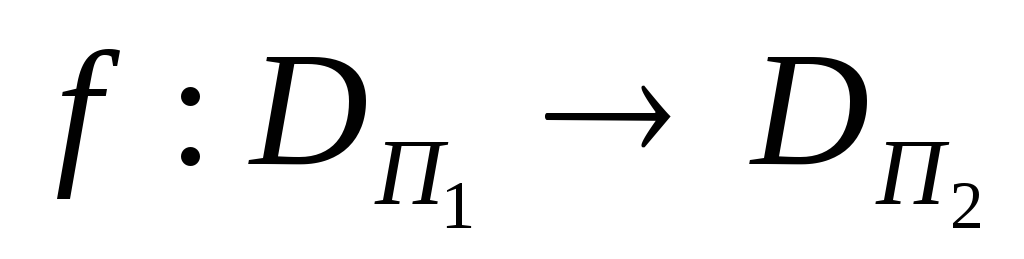
**Лемма 1:**

***Если L1  сводима к L 2, то ( также эквивалентно высказывание ).***

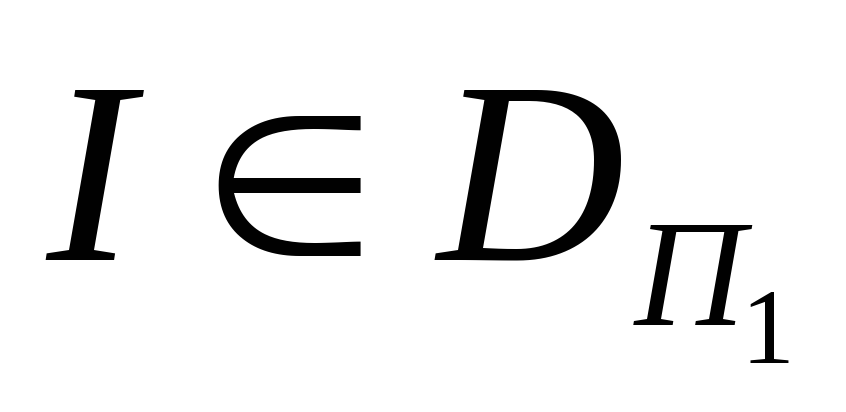
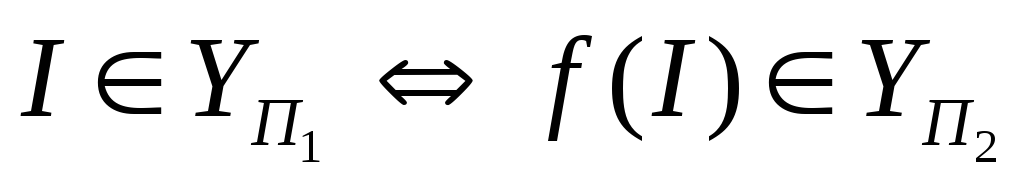
**Доказательство:**

**Пусть и– алфавиты языков *L1* и *L2*, а функцияосуществляет полиномиальную сводимость. Пусть M – ДМТ-программа вычисления *f*,*M2*– полиномиальная программа распознания *L2*. Полиномиальная программа распознавания*L1*может быть получена композицией программ M и M2. Ко входу применяем программу M для того, чтобы построить. Затем к *f(x)* применяется программа *M2*, которая выясняет справедливость отношения . Так как по определению сводимости, то описанная ДМТ-программа распознает язык *L1*. Полиномиальность этой программы следует из полиномиальности программ *M* и *M2*. Точнее, если *p* и *p2 –*полиномы, ограничивающие время работы программ *M* и *M2*, то , и, следовательно, время работы только что определённой программы ограничено функцией , которая является полиномом от длины *x***

**ч.т.д.**

**Если *П1* и *П2*– задачи распознавания,*e1*и*e2*– их схемы кодирования, то будем писать (относительно заданных схем кодирования), если существует полиномиальная сводимость языка к . На уровне задач полиномиальная сводимость задачи распознавания *П1* к задаче распознавания *П2* означает наличие функции, удовлетворяющей двум условиям:**

**1) *f* вычисляется полиномиальным алгоритмом.**

**2) Для всех ,.**

# Методы разработки трансляторов

## 17. Понятие формальной грамматики и ее основные атрибуты. Продукционное правило, вывод, сентенциальная форма, предложение, язык. Синтаксическое дерево и его связь с выводом.

1. Понятие формальной грамматики и ее основные атрибуты

Формальная грамматика — это система правил, определяющих конструкции, которые могут быть сгенерированы в языке. В контексте компьютерной науки, это часто используется для определения синтаксиса языка программирования или другого языка, подобного языку.

Основные атрибуты формальной грамматики:

* Набор терминалов (T), который представляет символы, из которых можно создавать строки в языке.
* Набор нетерминалов (N), символы, используемые для представления шаблонов или структур в языке.
* Набор продукций (P), каждая из которых представляет собой правило преобразования или замены.
* Стартовый символ (S), который является начальной точкой для генерации строк.

2. Продукционное правило, вывод, сентенциальная форма, предложение, язык

Продукционное правило - это правило, которое определяет, как одна последовательность символов может быть преобразована в другую.

Вывод - это процесс применения продукционных правил для преобразования начального символа в конечную строку.

Сентенциальная форма - это промежуточный результат в процессе вывода. Она может содержать как терминалы, так и нетерминалы.

Предложение - это конечная строка, полученная в результате процесса вывода, которая содержит только терминалы.

Язык - это набор всех возможных предложений, которые могут быть сгенерированы с использованием данной грамматики.

3. Синтаксическое дерево и его связь с выводом

Синтаксическое дерево, или дерево разбора, это дерево, которое представляет структуру предложения в соответствии с правилами грамматики.

Вершины дерева представляют символы предложения (как терминальные, так и нетерминальные), а ребра представляют применение продукционных правил. Процесс вывода может быть представлен в виде построения такого синтаксического дерева, где начальный символ представляет корень дерева, а конечное предложение - листья дерева.

## 18. Основные этапы трансляции и их особенности: лексический анализ, генерация машинного кода, синтаксический анализ. Виды синтаксического разбора.

Лексический анализ предшествует синтаксическому анализу и служит для перекодирования к виду, удобному для синтаксического.

Лексический анализ сводится к разбиению текста программы на лексические единицы (слова, лексемы) – конструкции, которые выступают в качестве терминальных символов для синтаксического анализа. Лексемы еще иногда называют символами или атомами.

Большинство лексем в языках программирования можно сгруппировать в следующие классы: − класс идентификаторов; − класс служебных слов; − класс констант (числовых или символьных); − класс операций (одно-, дву- или многолитерных); − класс разделителей (однолитерных или двулитерных). В лексическом анализе лексема представляется в виде пары (класс, значение). Значение может быть непосредственным (литеральным) или представлять собой ссылку на некоторую таблицу, в которой хранятся значения лексем.

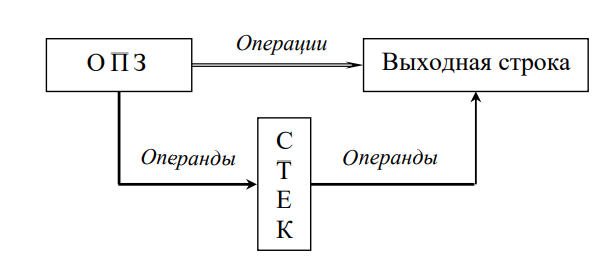
| Типы лексем | Обозначение |
| --- | --- |
| Служебные слова | W |
| Идентификаторы | I |
| Операторы | O |
| Разделители | R |
| Константы | N |
| C |

Основная функция лексического анализатора (сканера) состоит в выделении лексем из исходной программы и передаче их синтаксическому анализатору в некотором внутреннем представлении.

Далее необходимо перевести программу в обратную польскую запись для генерации машинного кода.

Перевод ОПЗ в текст на выходном (машинном) языке представляет собой следующий этап трансляции исходной программы в машинные коды. Для реализации этой процедуры также используется автомат с магазинной памятью (МП-автомат).

Эту процедуру можно схематично представить следующим образом:



Чтение символов операций из ОПЗ инициирует семантические процедуры, которые генерируют соответствующие заготовки машинного кода. Так же обрабатываются собранные в стеке операнды данной. Например, в случае обработки переменных и констант извлекается соответствующая им информация из таблиц идентификаторов и констант, которая используется для образования правильных адресных частей соответствующих машинных команд.

Основная задача синтаксического анализа - проверка исходной программы на соответствие грамматике языка программирования. Следует еще раз напомнить, что синтаксический анализ производится над кодом программы, который получен на выходе лексического анализа.

Результат синтаксического анализа, который часто называется разбором, представляется в виде дерева разбора. Данное дерево должно демонстрировать вывод исходной программы как цепочки символов из начального символа грамматики.

Для построения дерева разбора используются различные методы синтаксического анализа, каждый из которых имеет свои особенности.

1. Рекурсивный спуск - использует рекурсивные функции для обработки каждой структурной единицы языка. Используется для языков с контекстно-свободной грамматикой.
2. Метод нисходящего анализа - анализ начинается с корня дерева и стремится к листьям. Включает в себя рекурсивный спуск, LL-анализ и т. д.
3. Методы восходящего анализа (Bottom-Up Parsing): Эти методы начинают с токенов (терминальных символов) и строят синтаксическое дерево, двигаясь "снизу вверх", объединяя токены в более крупные структуры. К ним относятся LR-анализ, SLR-анализ, LALR-анализ и т. д.
4. Методы восходящего анализа (Bottom-Up Parsing): Эти методы начинают с токенов (терминальных символов) и строят синтаксическое дерево, двигаясь "снизу вверх", объединяя токены в более крупные структуры. К ним относятся LR-анализ, SLR-анализ, LALR-анализ и т. д.
5. И т.д.

# Криптографические протоколы

## 19. Понятие криптосистемы, классификация криптосистем, условие стойкости. Основная проблема симметричных шифров. Асимметричное шифрование: общий подход, примеры алгоритмов. Криптографические протоколы: распределение ключей, хеширование, электронная подпись.

**криптосистема** представляет собой набор [криптографических алгоритмов](https://translated.turbopages.org/proxy_u/en-ru.ru.bac91116-661bb627-5762e4c6-74722d776562/https/en.wikipedia.org/wiki/Cryptographic_algorithm?__ya_mt_enable_static_translations=1), необходимых для реализации определенной службы безопасности, такой как конфиденциальность ([шифрование](https://translated.turbopages.org/proxy_u/en-ru.ru.bac91116-661bb627-5762e4c6-74722d776562/https/en.wikipedia.org/wiki/Encryption?__ya_mt_enable_static_translations=1)).

Существует два типа криптосистем, в зависимости от того, как шифрование-дешифрование выполняется в системе:

1. Симметричное шифрование ключа
2. Асимметричное шифрование ключа

**Основная проблема симметричных шифров  
Проблема** состоит в том, что два одинаковых блока открытого текста превращаются в одинаковые блоки шифртекста. Это значит, что противник, имея только зашифрованный текст, может получать некоторую информацию об открытом тексте, следовательно, **шифр** является небезопасным.

Асимметричное шифрование — это **метод шифрования данных, предполагающий использование двух ключей — открытого и закрытого**. Открытый (публичный) ключ применяется для шифрования информации и может передаваться по незащищенным каналам. Закрытый (приватный) ключ применяется для расшифровки данных, зашифрованных открытым ключом.

Принцип действия асимметричного шифрования

Схема передачи данных между двумя субъектами (А и Б) с использованием открытого ключа выглядит следующим образом:

* Субъект А генерирует пару ключей, открытый и закрытый (публичный и приватный).
* Субъект А передает открытый ключ субъекту Б. Передача может осуществляться по незащищенным каналам.
* Субъект Б шифрует пакет данных при помощи полученного открытого ключа и передает его А. Передача может осуществляться по незащищенным каналам.
* Субъект А расшифровывает полученную от Б информацию при помощи секретного, закрытого ключа.

**Примерами** алгоритмов **асимметричного** **шифрования** являются алгоритмы **шифрования** RSA, Диффи-Хеллмана, Рабина и Эль-Гамаля.

**Протокол распределения ключей** (англ. Key distribution protocols) — это условленная последовательность действий пользователей (криптографический протокол) по созданию защищенного канала связи, заключающаяся в генерации и обмене сеансовыми ключами и аутентификации сообщений.

Основной задачей протоколов распределения ключей является выработка участниками (будем называть их в дальнейшем, как всегда, Алисой и Бобом) общего ключа.

Криптографическая хеш-функция — это алгоритм, который принимает на вход сообщение и превращает его в уникальный битовый массив фиксированного размера. Такой массив называется хешем или хеш-суммой, а сам процесс — хешированием.

Для каждого сообщения алгоритм создаёт свой уникальный хеш. Если пропустить одно и то же сообщение через алгоритм, хеш на выходе будет неизменным. Но если заменить в исходных данных хотя бы одну букву, хеш изменится до неузнаваемости.

Хеш-функция используется для разных задач:

* для безопасного хранения паролей на сайте;
* для создания цифровых подписей;
* для защиты игровых данных;
* для подтверждения транзакций в блокчейне и многого другого.

Протоколы электронной цифровой подписи (эцп)

В основе протокола этого класса содержится некоторый алгоритм вычисления ЭЦП на передаче с помощью секретного ключа отправителя и проверки ЭЦП на приёме с помощью соответствующего открытого ключа, извлекаемого из открытого справочника, но защищенного от модификаций. В случае положительного результата проверки протокол, обычно, завершается операцией архивирования принятого сообщения, его ЭЦП и соответствующего открытого ключа. Операция архивирования может не выполняться, если ЭЦП используется только для обеспечения свойств целостности и аутентичности принятого сообщения, но не безотказности. В этом случае, после проверки, ЭЦП может быть уничтожена сразу или по прошествии ограниченного промежутка времени ожидания.

## 20. Симметричное шифрование. Блочные шифры: шифры Фейстеля и шифры не – Фейстеля, примеры. Поточные шифры: ПСП, примеры шифров, режимы работы DES и ГОСТ.

При шифровании таким методом ключ, используемый для зашифровки данных, совпадает с ключом для их расшифровки.

Для работы применяется всего один пароль. Происходит всѐ следующим образом:

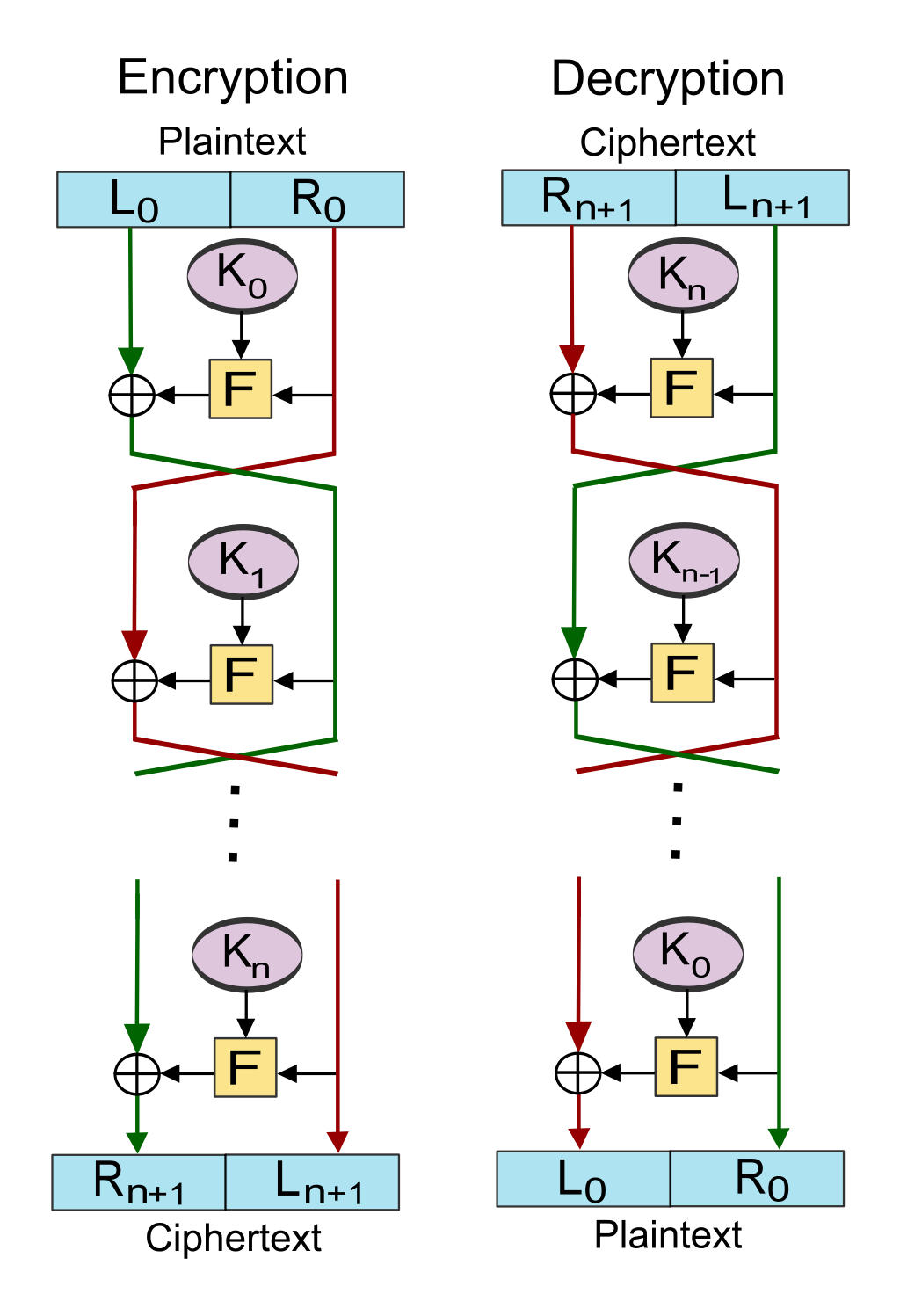
1. Существует некий математический алгоритм шифрования.
2. На его вход подаѐтся текст и пароль.
3. На выходе получаем зашифрованный текст.
4. Если хотим получить исходный текст, применяется тот же самый пароль, но с алгоритмом дешифрования.

Симметричный алгоритм прекрасно подходит при передаче больших объѐмов зашифрованных данных.

**Бло́чный шифр** — разновидность симметричного шифра, оперирующего группами бит фиксированной длины — блоками. От поточных шифров работа блочного отличается обработкой бит группами, а не потоком. При этом блочные шифры медленнее поточных. К достоинствам блочных шифров относят сходство процедур шифрования и расшифрования, которые, как правило, отличаются лишь порядком действий. Это упрощает создание устройств шифрования, так как позволяет использовать одни и те же блоки в цепях шифрования и расшифрования.

Блочный шифр состоит из двух парных алгоритмов: шифрования и расшифрования. Оба алгоритма можно представить в виде функций. Функция шифрования E на вход получает блок данных M размером n бит и ключ K размером k бит и на выходе отдает блок шифротекста C размером n бит. Функция расшифрования D на вход получает шифротекст C, ключ K и на выходе отдает M.

**Шифр Фе́йстеля** — один из методов построения блочных шифров. Шифр состоит из ячеек, называемых ячейками Фейстеля. На вход каждой ячейки поступают данные и ключ. На выходе каждой ячейки получают изменённые данные и изменённый ключ. Все ячейки однотипны. Ключ выбирается в зависимости от алгоритма шифрования/расшифрования и меняется при переходе от одной ячейки к другой. При шифровании и расшифровании выполняются одни и те же операции, отличается только порядок ключей.



*Шифрование:*

1. На вход поступает блок данных. Данный блок делится на две равные части - левый и правый подблоки.
2. Правый подблок изменяется функцией с использованием раундового ключа.
3. Результат складывается по модулю 2 с левым подблоком.
4. В следующем раунде результат будет использоваться в роли правого подблока, а текущий правый подблок (в неизмененном виде) будет использоваться в роли левого.
5. По какому-либо математическому правилу вычисляется новый раундовый ключ.

Перечисленные операции повторяются количество раз, равное количеству раундов.

Расшифрование происходит аналогично, но раундовые ключи идут в обратном порядке.

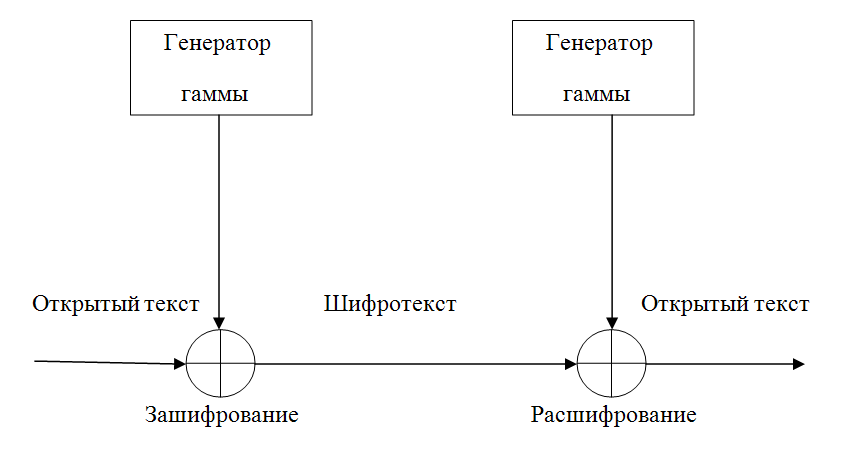
В качестве функции, могут использоваться:

* S-блок - состоит из дешифратора (преобразователь n-разрядного двоичного сигнала в одноразрядный сигнал по основанию 2^n), коммутатора, шифратора (операция, обратная дешифратору)
* P-блок - изменяет положение цифр.
* Циклический сдвиг
* Сложение по модулю n
* Умножение по модулю n



**Пото́чный шифр** — это симметричный шифр, в котором каждый символ открытого текста преобразуется в символ шифрованного текста в зависимости не только от используемого ключа, но и от его расположения в потоке открытого текста.

Простейшая реализация поточного шифра изображена на рисунке ниже. Генератор гаммы выдает ключевой поток. Поток битов шифротекста получается с помощью применения операции XOR над битами открытого текста и битами ключевого потока.



Практически во всех каналах передачи данных для потоковых систем шифрования присутствуют помехи. Поэтому для предотвращения потери информации решают проблему синхронизации шифрования и расшифрования текста. По способу решения этой проблемы шифросистемы подразделяются на синхронные и системы с самосинхронизацией.

*Синхронные потоковые шифры* — шифры, в которых поток ключей генерируется независимо от открытого текста и шифротекста.

При шифровании генератор потока ключей выдаёт биты потока ключей, которые идентичны битам потока ключей при дешифровании. Потеря знака шифротекста приведёт к нарушению синхронизации между этими двумя генераторами и невозможности расшифрования оставшейся части сообщения.

Обычно синхронизация производится вставкой в передаваемое сообщение специальных маркеров. В результате этого пропущенный при передаче знак приводит к неверному расшифрованию лишь до тех пор, пока не будет принят один из маркеров.

*Самосинхронизирующиеся потоковые шифры* — шифры, в которых ключевой поток создаётся функцией ключа и фиксированного числа знаков шифротекста.

Внутреннее состояние генератора потока ключей является функцией предыдущих N битов шифротекста. Поэтому расшифрующий генератор потока ключей, приняв N битов, автоматически синхронизируется с шифрующим генератором.

Реализация этого режима происходит следующим образом: каждое сообщение начинается случайным заголовком длиной N битов; заголовок шифруется, передаётся и расшифровывается; расшифровка является неправильной, зато после этих N бит оба генератора будут синхронизированы.

# Распределенные задачи и алгоритмы

## 21. Дайте определение распределенной системы (DS). Перечислить требования к DS. Перечислите и назовите функции логических программных слоев распределенной системы (DS).

Распределенная система – набор независимых компьютеров, не имеющих общей совместно используемой памяти и общего единого времени (таймера) и взаимодействующих через коммуникационную сеть посредством передачи сообщений, где каждый компьютер использует свою собственную оперативную память и на котором выполняется отдельный экземпляр своей операционной системы, предоставляя свои службы друг другу для решения общей задачи.

Требования:

Прозрачность - Способность скрывать свою распределенную природу, а именно, распределение процессов и ресурсов по множеству компьютеров, и представляться для пользователей и разработчиков приложений в виде единой централизованной компьютерной системы.

открытость:

1. Гарантирует расширяемость
2. lМобильность приложений
3. lИнтероперабельность
4. lМобильность пользователя
5. lВозможность повторного использования

Безопасность - посылка значимой информации по сети безопасно и эффективно

Масштабируемость:

·Нагрузочная масштабируемость - способность системы увеличивать свою производительность при увеличении нагрузки путем замены существующих аппаратных компонентов на более мощные или путем добавления новых аппаратных средств.

·увеличение производительности каждого компонента системы с целью повышения общей производительности называют вертикальным масштабированием

·увеличение количества сетевых компьютеров (серверов) распределенной системы – горизонтальным масштабированием.

· Географическая масштабируемость - способность системы сохранять свои основные характеристики, такие как производительность, простота и удобство использования, при территориальном разнесении ее компонентов от более локального взаиморасположения до более распределенного.

Административная масштабируемость характеризует простоту управления системой при увеличении количества административно независимых организаций, обслуживающих части одной распред. системы

## 22. Поясните разницу между физическим и логическим временем, что такое парадоксы времени в распределенных системах, привести примеры. В чем различие требований к синхронным и асинхронным системам. Задача о двух генералах.

**Физическое время** - реальное время, проходящее с момента начала какого-либо события до его завершения. Определяется внешними факторами, такими как характеристики аппаратного обеспечения.

**Логическое время** - абстрактное понятие, которое используется для упорядочивания событий в распределенных системах. Логическое время основано на порядке событий, как они воспринимаются системой, а не на реальном прошедшем времени.

**Парадокс последовательности событий:** В распределенных системах порядок событий может быть разным на разных узлах из-за асинхронной передачи данных. Это может привести к неправильному восприятию порядка событий и несоответствию логического и физического времени.

**Требования к синхронным/асинхронным системам**

В синхронных системах все узлы имеют общий и точно синхронизированный источник времени. Синхронные системы обеспечивают более простое управление временем и обменом сообщениями, но они требуют более высокой степени надежности и стабильности сетевых соединений.

**В асинхронных** системах отсутствует общий источник времени, и узлы могут функционировать независимо друг от друга. Асинхронные системы обеспечивают более гибкую архитектуру и могут быть более устойчивыми к сбоям, но требуют более сложных алгоритмов синхронизации и управления временем.

**Задача о двух генералах** - классическая проблема в области распределенных систем, иллюстрирующая сложности достижения согласованности в асинхронных сетях. Два военачальника, каждый со своей армией, стоят перед городом и планируют атаку. Они должны согласовать временные рамки атаки, отправив сообщения друг другу через ненадежную связь. Однако из-за неопределенности времени доставки сообщений и возможности их потери они сталкиваются с проблемой, когда невозможно гарантировать, что они получили информацию о времени атаки вовремя и одновременно.

## 23. Назовите способы синхронизации в распределённых системах. Сопоставьте алгоритмы Беркли и Кристиана. Перечислите задачи и условности работы алгоритма Лэмпорта.

1. Распределенная блокировка - механизм блокировки для доступа к разделяемым ресурсам.
2. Кворумы - способ голосования в распределенной системе, когда принятие решения зависит от определенного количества участников.
3. Алгоритмы синхронизации времени - позволяют синхронизировать временные метки на разных узлах распределенной системы. Физическое и логическое время

Алгоритмы Беркли и Кристиана относятся к алгоритмам синхронизации времени. Алгоритм Беркли использует мастер-узел для сбора и анализа временных данных со всех узлов, после чего корректирует время на каждом узле в соответствии со средним значением временных данных. Алгоритм Кристиана работает похожим образом, но вместо мастер-узла используется узел-сервер, который отсылает свое значение времени на все другие узлы, и каждый узел корректирует свое время в соответствии с этим значением.

Алгоритм Лэмпорта - это алгоритм синхронизации времени, который используется для решения задачи взаимного исключения в распределенных системах. Этот алгоритм позволяет каждому узлу выдать локальный номер запроса и синхронизировать номера на всех узлах, чтобы определить порядок выполнения операций и контролировать доступ к разделяемым ресурсам.

Основной идеей алгоритма Лэмпорта является использование логических часов. Каждый узел самостоятельно генерирует локальный номер запроса (события) и записывает его в свои логические часы. Логические часы состоят из двух частей: счетчика и идентификатора узла. Счетчик увеличивается каждый раз при создании нового события, а идентификатор узла определяет, на каком узле это событие было создано.

Когда узел хочет получить разделенный ресурс, он отправляет запрос на все другие узлы, содержащие логические часы с номером события. Каждый узел отвечает на запрос, включая свой локальный номер события в сообщение. Получатель сравнивает номера событий и устанавливает, кто должен получить доступ к разделяемому ресурсу первым. Если не может быть установлен порядок выполнения запрошенных операций на разных узлах, сервер решает, кто получит доступ к ресурсу первым.

Алгоритм работает при условии, что существует канал связи между каждой парой узлов, в том числе в обоих направлениях. Алгоритм Лэмпорта выполняет следующие задачи:

Контролирует порядок выполнения операций на разных узлах.

Обеспечивает возможность совместного доступа к разделяемым ресурсам.

Гарантирует, что каждый узел работает в правильном порядке и исполняет каждую операцию только один раз

## 24. Централизованные и децентрализованные алгоритмы доступа к общим ресурсам в распределенным системам. Алгоритмы взаимного исключения, изложить основную идею их работы, перечислить разновидности, назвать различия.

Основная идея механизм взаимного исключения выражается в удовлетворении основного требования: во время выполнения некоторой критической секции кода одного процесса с обращениями к разделяемым ресурсам ни один другой процесс не должен выполняться в своей критической секции, работающей с любым из этих ресурсов. То есть во время работы в КС только один процесс имеет право с ним работать, остальные должны подождать.

(Доп. инфа) Распределенные алгоритмы взаимного исключения должны опираться исключительно на обмен сообщениями между процессами. Разработка таких алгоритмов осложняется тем, что приходится иметь дело с произвольными задержками передачи сообщений и отсутствием полной информации о состоянии всей системы.

Основные требования к алгоритмам взаимного исключения:  
-Безопасность. В каждый момент времени в КС может выполняться не более одного процесса.

-Живучесть. Каждый запрос на вход в КС рано или поздно должен быть удовлетворен (напр., не должна быть ситуация взаимной блокировки)

Все многообразие алгоритмов взаимного исключения в распределенных системах базируется на реализации одного из двух основных принципов:

-На основе получения разрешений. В этом случае для входа в КС процессу требуется собрать "достаточное" число разрешений от других процессов распределенной системы.  
(Доп. инфа) Безопасность - число разрешений сможет получить лишь один процесс из всех процессов, желающих войти в КС.  
Живучесть - упорядочивание запросов по логическому времени или с помощью ациклического графа предшествования.

-На основе передачи маркера. Право использование КС материализуется в виде некого маркера, который в каждый момент времени может содержаться только у одного процесса или же находиться в канале в состоянии пересылки от одного процесса к другому.

(Доп. инфа) Безопасность - гарантирована ввиду уникальности маркера.  
Живучесть - непрерывное перемещение маркера среди всех процессов распределенной системы или перемещение маркера лишь в ответ на получение запроса от заинтересованного в нем процесса.

Централизованные алгоритмы всегда асимметричны, так как помимо обычных процессов имеют главный сервер, который и занимается либо выдачей маркера или разрешений подконтрольным процессам.

Децентрализованные алгоритмы симметричны, так как каждый процесс имеет одинаковые права в голосовании при выдаче разрешений, а также в запросе на получение маркера. Примерами являются ***Алгоритм Лэмпорта (на основе разрешений) и Широковещательный алгоритм Сузуки-Касами (на основе маркера.***

## 25. Понятие процесса. Состояния процесса. Операции над процессами. Process Control Block и контекст процесса. Алгоритмы планирования. First-Come, First-Served (FCFS); Round Robin (RR).

Процесс - это программа или команда, выполняемая на компьютере. Операционная система может одновременно выполнять множество различных процессов. Следует понимать, что процесс может состоять из нескольких потоков, обратное не верно.

Компьютерная программа сама по себе это только пассивная совокупность инструкций, в то время как процесс — это непосредственное выполнение этих инструкций.

Часто процессом называют выполняющуюся программу и все её элементы: адресное пространство, глобальные переменные, регистры, стек, открытые файлы и т. д.

Каждый процесс может находиться как минимум в двух состояниях: процесс исполняется и процесс не исполняется.



Процесс, находящийся в состоянии процесс исполняется, может через некоторое время завершиться или быть приостановлен операционной системой и снова переведен в состояние процесс не исполняется. Приостановка процесса происходит по одной из двух причин: для его дальнейшей работы потребовалось возникновение какого-либо события или истек временной интервал, отведенный операционной системой для работы этого процесса. После этого операционная система по определенному алгоритму выбирает для исполнения один из процессов, находящихся в состоянии процесс не исполняется, и переводит его в состояние процесс исполняется. Новый процесс, появляющийся в системе, первоначально помещается в состояние процесс не исполняется.

В этой связи подсистема *Управление процессом* должна выполнять следующие операции над процессами:

1. Создание.
2. Уничтожение.
3. Запуск.
4. Блокировка.
5. Приостановка.
6. Возобновление, а также
7. Изменение диспетчерского приоритета процесса (работа с очередями).

Блок управления процессами(The Process Control Block) - это фундаментальная структура данных для управления процессами в операционной системе. Он позволяет операционной системе переключаться между процессами, распределять ресурсы, управлять прерываниями и поддерживать общую стабильность и производительность системы.

Контекст процесса включает в себя содержимое адресного пространства задачи, выделенного процессу, а также содержимое относящихся к процессу аппаратных регистров и структур данных ядра.

Существует шесть популярных алгоритмов планирования

* Планирование «первым пришел — первым обслужен» (FCFS)
* Планирование Shortest-Job-Next (SJN)
* Приоритетное планирование
* Самое короткое оставшееся время
* Круглый Робин (RR) Планирование
* Планирование многоуровневых очередей

## **First Come First Serve (FCFS)**

В алгоритме First Come First Serve Algorithm мы позволяем процессам выполняться линейно. Это означает, что тот процесс, который первым попадет в очередь готовности, будет выполнен первым. Это показывает, что алгоритм First Come First Serve следует принципу First In First Out (FIFO).

* Задания выполняются в порядке поступления.
* Это не упреждающий алгоритм планирования.
* Легко понять и реализовать.
* Его реализация основана на очереди FIFO.
* Низкая производительность, так как среднее время ожидания велико.

## **Круглый Робин**

* Round Robin — это алгоритм планирования вытесняющих процессов.
* Каждому процессу предоставляется определенное время для выполнения, оно называется **квантом** .
* Как только процесс выполняется в течение заданного периода времени, он прерывается, и другой процесс выполняется в течение заданного периода времени.
* Переключение контекста используется для сохранения состояний вытесненных процессов.

Round Robin — это алгоритм планирования вытесняющих процессов.

Каждому процессу предоставляется определенное время для выполнения, оно называется **квантом** .

Как только процесс выполняется в течение заданного периода времени, он прерывается, и другой процесс выполняется в течение заданного периода времени.

# Управление информацией

## 26. Реляционная модель данных. Отношения и их свойства. Ограничения целостности. Ключи. Первичный ключ. Альтернативный ключ. Суррогатный ключ. Внешний ключ.

Реляционная модель представляет собой совокупность данных, состоящую из набора двумерных таблиц. В теории множеств таблице соответствует термин *отношение*.

Отношение – это подмножества декартового произведение множеств A и B. Так как отношение является множеством, то элементы в нём, и, соответственно, строки в таблице, повторяться не могут.

Ограничение целостности - это некоторое утверждение, которое может быть истинным или ложным в зависимости от состояния базы данных.

База данных находится в согласованном (целостном) состоянии, если выполнены все ограничения целостности, определенные для базы данных.

По области действия ограничения делятся на:

- ограничения домена;

- ограничения атрибута;

- ограничения кортежа;

- ограничения отношения (целостность сущностей);

- ограничения базы данных (целостность по ссылкам).

Требование целостности по ссылкам, или требование внешнего ключа состоит в том, что для каждого значения внешнего ключа, появляющегося в дочернем отношении, в родительском отношении, на которое ведет ссылка, должен найтись кортеж с таким же значением первичного ключа, либо значение внешнего ключа должно быть неопределенным.

Первичный ключ — это поле или набор полей со значениями, которые являются уникальными для всей таблицы.

Альтернативный ключ — это потенциальный ключ, не ставший первичным.

Суррогатный ключ - это генерируемое системой значение, не имеющее смысла в терминах предметной области и которое используется для однозначной идентификации записи в таблице.

# Модели интеллектуальных систем

## 27. Предикатные модели представления знаний (атомарная, предикатная, с исключениями из правил, иерархическая, сценарная).

АТОМАРНАЯ:

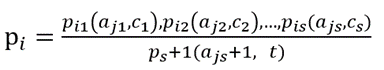
**Атомы** - высказывания в заданной области деятельности, которые в каждой конкретной ситуации могут быть либо истинными, либо ложными (менять значение не могут).

**Знания, относящиеся к атомам, образуют продукции**: **π**=(a1…an)/an+1, где a1…an - посылки, an+1 - заключение. Продукция , для которой n=0, называется аксиомой.

Если все a1…an истинны, то an+1 истинно. Если что-то в a1…an ложно, то это не значит, что an+1 ложно. Оно недоказуемо. **Продукционной системой** наз-ся всякое конечное мн-во атомарных продукций. Из них аксиомы – это всегда начальные данные.

ПРЕДИКАТНАЯ

Пусть дана предметная область, которая оперирует параметрами объектов a1, …, an, принимающих значения из множеств q1, …, qn. Для этих множеств определено множество предикатов p1(x, y), …, pn(x, y) [x - значение из a1..an, y - значение из q1..qn].

**Предикатная продукция**:  pi1…pis – посылки, t – терм – выражение, составленное из операций, имен и констант.

Предикатная интерпретация применения продукций состоит в том, чтобы все предикаты посылок продукций были истинными, а для терма должны быть известны значения всех параметров.

С ИСКЛЮЧЕНИЯМИ ИЗ ПРАВИЛ

Пример:  

Определение: пара продукций  и 

образуют пару правила-исключ-е из правила, если выполним-ь усл-й исключ- блокирует прим-е основного правила.У исключ-й из правила могут быть свои исключ-я. Такая система образует конструкцию, кот-я назыв-я гнездом продукций.

ИЕРАРХИЧЕСКАЯ

Картинка: корень (пустой), и от него R1 ,R2 ,…,Rk . Это разные наборы условий. Каждая Ri разбивается на Di,1,Di,2 ,…, Di,j ,…Di,mi . Di, j разбивается на Ai,j 1(это верхний индекс у всех далее) →Bi,j 1 , Ai,j 2 →Bi,j 2, … Ai,j s →Bi,j s . Под каждым ядром продукции идёт: Ni,j 1 , Ni,j 2 ,…,Ni,j s. Где-то могут быть одинаковые элементы (например, Di,1=Di,2 ).

Иерархическая модель является способом организации создания баз знаний, когда база составляется из нескольких независимых частей, имеющих иерархическую структуру, и каждая часть может быть сформирована при участии экспертов, если она небольшая. В основном эксперты приписывают структуру базы знаний, как процесс составления этой базы знаний.

Формирование ядер продукции осуществляется для узких ситуаций, где возможно описание всех знаний одновременно. Иерархическая модель хороша для формирования баз знаний, управления процессов.

СЦЕНАРНЫЕ МОДЕЛИ  
Рассмотрим модель базы продукции которой связаны с этапами процесса связывания задач данной системы.

Этапы решения задач выстраиваются в последовательности, формирующие диаграммы. Если два этапа, каждый из которых управляет базой знаний: **описание мн-ств объектов составляющих результаты выполнения этапа (**предикаты завершения работы ) и **условие перехода к следующему этапу**.

При завершении 1 го этапа и выполнимости предиката Т выполняется перенос содержимого результата в базу Б2.

1(P1)(B1)(Q1) T-> 2(P2)(B2)(Q2)

В общем случае этапы составляют сценарии, которые могут быть последовательными, или параллельным или вариантным.

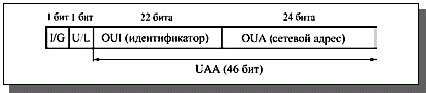
# Компьютерные сети

## 28. Перечислить виды адресации в сетях, дать краткое описание, на каких уровнях модели OSI используются. Перечислить и охарактеризовать алгоритмы доступа к среде передачи данных в компьютерных сетях.

Адресация в сетях проходит несколько этапов преобразования для установления однозначного соответствия адреса приемника приемнику, адреса источника источнику. Это преобразование производится соответствующими уровнями модели OSI.

На верхних уровнях модели OSI используются специальные имена, например, идентификаторы процессов, рабочих станций или символьные доменные имена; на сетевом уровне используется сетевой, например, IP-адрес; на канальном – физические адреса устройств (MAC-адреса).

Подуровень MAC канального уровня модели OSI работает с физическими адресами, которые называются МАС-адресами. Они применяются в сетях Ethernet, Fast Ethernet, Token-Ring, FDDI, 100VG-AnyLAN и представляют собой 12 шестнадцатеричных цифр (48 бит), записанных в микросхему сетевого адаптера (например, 17:A4:2C:43:2F:09).



Для широковещательной передачи (то есть передачи всем абонентам сети одновременно) применяется специально выделенный сетевой адрес, все 48 битов которого установлены в единицу. Его принимают все абоненты сети независимо от их индивидуальных и групповых адресов.

**Система адресации ip V.4**

Адреса сетевого уровня IP назначаются и используются операционной системой. У работающей ОС может быть один и более IP-адресов. Если используется несколько интерфейсных карт, то ОС должна назначить однозначное соответствие каждого IP-адреса конкретному порту сетевого обмена, например, сетевому адаптеру.

Механизм адресации IP v.4 предполагает деление адреса на 2 части: № сети и № узла.

Классу **A** соответствует диапазон адресов 1.0.0.0 – 127.255.255.255.

Классу **B** соответствует диапазон адресов 128.0.0.0 – 191.255.255.255.

Классу **С** соответствует диапазон адресов 192.0.0.0 – 223.255.255.255.

Классу **D** соответствует диапазон адресов 224.0.0.0 – 239.255.255.255.

Классу **E** соответствует диапазон адресов 240.0.0.0 – 247.255.255.255.

Класс **А** используется для самых крупных сетей, насчитывающих до 16677216 узлов. Таких сетей максимальное количество 126. В классе **В** максимально – 65536 узлов, количество сетей – 2142. Класс **С** оперирует количеством адресов, не превышающим 254 узла. Это наиболее распространенный и эффективный класс адресов.

***Бесклассовая адресация*** (**Classless InterDomain Routing – CIDR**) – метод IP-адресации, позволяющий гибко управлять пространством IP-адресов, не используя жёсткие рамки классовой адресации. Способ основывается на переменной длине маски подсети (Variable Lenght Subnet Mask – VLSM), в то время как в классовой адресации длина маски строго фиксирована 0,1,2 или 3 установленными байтами.

Помимо этих адресов существует набор специализированных (отладочных) адресов. Например: 127.0.0.0, 127.0.0.1 и некоторые другие адреса, начинающиеся со 127. Такие адреса принято называть Loopback. Данные не передаются по сети, а передаются модулям верхнего уровня узла-отправителя как только что принятые.

**Система адресации ip V.6**

Свойства адресации IPv.6:

* 128-разрядная адресация;
* связь одного адреса с несколькими интерфейсами;
* автоматическое назначение адреса и CIDR-адресация.

В IPv.6 определяется несколько типов адресов (Они отличаются по префиксу формата– 10, 110, 1110, …):

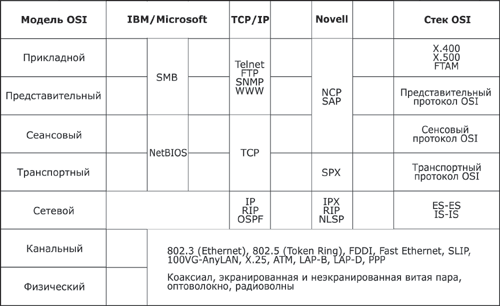
– UniCast – уникальный адрес.

– MultiCast – групповой адрес.

– AnyCast (в некоторых источниках – Claster) – это тип адреса, при котором пакет доставляется ближайшему узлу из группы интерфейсов, имеющих этот адрес.

## 29. Стандартные сетевые протоколы (в пределах изученных стеков протоколов). Перечислить, поставить в соответствие модели OSI. Сетевые и транспортные протоколы стека TCP/IP. Перечислить, назвать основные функции. Алгоритм скользящего окна. Основные параметры и характеристики.

Соответствие протоколов модели OSI:



**Стек TCP/IP.**

Стек TCP/IP – это набор иерархически упорядоченных сетевых протоколов. Название стек получил по двум важнейшим протоколам – TCP (Transmission Control Protocol) и IP (Internet Protocol). Помимо них в стек входят ещё несколько десятков различных протоколов. В настоящее время протоколы TCP/IP являются основными для Интернета, а также для большинства корпоративных и локальных сетей.

В операционной системе Microsoft Windows Server 2003 стек TCP/IP выбран в качестве основного, хотя поддерживаются и другие протоколы (например, стек IPX/SPX, протокол NetBIOS).

Стек протоколов TCP/IP обладает двумя важными свойствами:

* платформонезависимостью, т. е. возможна его реализация на самых разных операционных системах и процессорах;
* открытостью, т. е. стандарты, по которым строится стек TCP/IP, доступны любому желающему.

**Обзор основных протоколов.**

***Протокол IP*** *(Internet Protocol) –* это основной протокол сетевого уровня, отвечающий за адресацию в составных сетях и передачу пакета между сетями. Протокол IP является *дейтаграммным* протоколом, т. е. не гарантирует доставку пакетов до узла назначения. Обеспечением гарантий занимается протокол транспортного уровня TCP.

***Протоколы RIP*** *(Routing Information Protocol –* протокол маршрутной информации*) и OSPF (Open Shortest Path First – «*первыми открываются кратчайшие маршруты»*)* – протоколы маршрутизации в IP-сетях.

***Протокол ICMP*** *(Internet Control Message Protocol –* протокол управляющих сообщений в составных сетях) предназначен для обмена информацией об ошибках между маршрутизаторами сети и узлом-источником пакета. С помощью специальных пакетов сообщает о невозможности доставки пакета, о продолжительности сборки пакета из фрагментов, об аномальных величинах параметров, об изменении маршрута пересылки и типа обслуживания, о состоянии системы и т. п.

***Протокол ARP*** *(Address Resolution Protocol* – протокол преобразования адресов) преобразует IP-адреса в аппаратные адреса локальных сетей. Обратное преобразование осуществляется с помощью протокола *RAPR* (Reverse ARP).

Далее рассматриваются протоколы прикладного уровня.

***HTTP*** *(HyperText Transfer Protocol* – протокол передачи гипертекста) – протокол доставки web-документов, основной протокол службы WWW.

***FTP*** *(File Transfer Protocol* – протокол передачи файлов) – протокол для пересылки информации, хранящейся в файлах.

***POP3*** *(Post Office Protocol version 3* – протокол почтового офиса) и ***SMTP*** *(Simple Mail Transfer Protocol* – простой протокол пересылки почты) – протоколы для доставки входящей электронной почты (POP3) и отправки исходящей (SMTP).

***Telnet***– протокол эмуляции терминала, позволяющий пользователю подключаться к другим удалённым станциям и работать с ними со своей машины, как если бы она была их удалённым терминалом.

***SNMP*** *(Simple Network Management Protocol* – простой протокол управления сетью) предназначен для диагностики работоспособности различных устройств сети.

Протоколы транспортного уровня могут решать проблему негарантированной доставки сообщений («дошло ли сообщение до адресата?»), а также гарантировать правильную последовательность прихода данных. В стеке TCP/IP транспортные протоколы определяют, для какого именно приложения предназначены эти данные. Два главных транспортных протокола - TCP и UDP.

* **TCP** - «гарантированный» транспортный механизм с предварительным установлением соединения, предоставляющий приложению надёжный поток данных, дающий уверенность в безошибочности получаемых данных, перезапрашивающий данные в случае потери и устраняющий дублирование данных. TCP позволяет регулировать нагрузку на сеть, а также уменьшать время ожидания данных при передаче на большие расстояния. Более того, TCP гарантирует, что полученные данные были отправлены точно в такой же последовательности. В этом его главное отличие от UDP.
* **UDP** - протокол передачи датаграмм без установления соединения. Также его называют протоколом «ненадёжной» передачи, в смысле невозможности удостовериться в доставке сообщения адресату, а также возможного перемешивания пакетов. В приложениях, требующих гарантированной передачи данных, используется протокол TCP.

На транспортном уровне появляется понятие «порт». Для передачи данных между процессами устройства (например, передача сообщения из одного приложения «вконтакте» другому приложению «вконтакте») недостаточно знать только IP-адрес и его маску. Ещё необходимо знать, какому процессу адресован пакет данных.

Для передачи пакетов между процессами было принято выделить на каждом хосте виртуальные порты, которые могут прослушиваться процессами.

**Порт** - это число от 0 до 65535, которое присваивается процессу на время. Порт чем-то похож на кабинку, которую можно занять определённой программе (ну а совокупность портов можно представить как гостиницу).

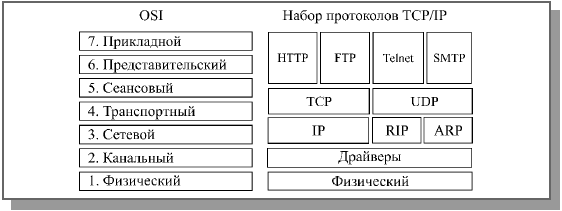
Порт порт также может определять, какой протокол используется на прикладном уровне.

Порты выделяются как для TCP, так и для UDP (т.е. порты для одного протокола не пересекаются с портами другого).

Прикладной уровень предназначен для работы сетевых приложений. Для обмена информацией они используют свои протоколы, например: протокол HTTP для обмена гипертекстовой разметкой, протокол FTP (передача файлов), SMTP (электронная почта), SSH (безопасное соединение с удалённой машиной), DNS (преобразование символьных имён в IP-адреса) и многие другие.

В массе своей эти протоколы работают поверх TCP или UDP и привязаны к определённому порту, например:

* HTTP на TCP-порт 80 или 8080;
* FTP на TCP-порт 20 (для передачи данных) и 21 (для управляющих команд);
* SSH на TCP-порт 22;
* запросы DNS на порт UDP (реже TCP) 53.



**Реализация скользящего окна в протоколе TCP**

Особенность использования алгоритма скользящего окна в протоколе TCP состоит в том, что, хотя единицей передаваемых данных является сегмент, окно определено на множестве нумерованных байтов неструктурированного потока данных, поступающих с верхнего уровня и буферизуемых протоколом TCP. Получающий модуль TCP отправляет «окно» посылающему модулю TCP. Данное окно задает количество байтов (начиная с номера байта, о котором уже была выслана квитанция), которое принимающий модуль TCP готов в настоящий момент принять.

Квитанция (подтверждение) посылается только в случае правильного приема данных. Отсутствие квитанции означает либо прием искаженного сегмента, либо потерю сегмента, либо потерю квитанции. В качестве квитанции получатель сегмента отсылает ответное сообщение (сегмент), в которое помещает число, на единицу превышающее максимальный номер байта в полученном сегменте. Это число часто называют номером очереди.



Если размер окна равен W, а последняя по времени квитанция содержала значение N, то отправитель может посылать новые сегменты до тех пор, пока в очередной сегмент не попадет байт с номером N+W. Этот сегмент выходит за рамки окна, и передачу в таком случае необходимо приостановить до прихода следующей квитанции.

# Обработка больших данных

## 30. Основные понятия статистики и дескриптивный анализ. Генеральная совокупность и выборка. Меры центральной тенденции, их сравнительный анализ. Опишите стандарты жизненного цикла Big Data. Приведите классификацию методов Data Mining.

Статистические данные - данные, полученные в результате обследования большого числа объектов или явлений (то есть, математическая статистика имеет дело с массовыми явлениями).

Предметом изучения в статистике являются изменяющиеся (варьирующие) признаки  
 Статистическая совокупность — это результат описания или измерения общих признаков объектов исследования.

Выборка представляет собой случайным образом отобранную часть популяции (**генеральной совокупности**)

Выборка должна быть репрезентативной - её свойства должны отражать свойства популяции.

Для этого она должна быть случайной (random) – все особи в популяции должны иметь одинаковые шансы попасть в неё, и попадание в выборку одного элемента не должно влиять на попадание другого элемента.

Частотное распределение переменной (frequency distribution) – это соответствие между значениями переменной и их вероятностями (на практике – количеством таких значений в выборке)

Вариационный ряд – ряд, в котором сопоставлены (по степени возрастания или убывания) варианты и соответствующие им частоты.

Вариантами считаются отдельные значения признака, которые он принимает в вариационном ряду

Частота число, показывающее, сколько раз повторяется варианта.

Частостью или относительной частотой – называется отношение частоты к объему выборки (n).

Гистограмма – графическое представление частотного распределения, разбитого по интервалам, где высота столбика отражает ЧАСТОТУ

Полигон используют для дискретных рядов. На оси абсцисс в произвольном масштабе откладывают значения аргумента, а на оси ординат - значения частот или относительных частот.

Кумулята служит для графического изображения кумулятивного (с накоплением) вариационного ряда

**Меры центральной тенденции распределения:**

1. Среднее значение – сумма всех значений переменной, делённая на количество значений
2. Медиана (median) – значение, которое делит распределение пополам (его площадь в т.ч.): половина значений больше медианы, половина – меньше

Квартили (quartiles) делят распределение на четыре части так, что в каждой из них оказывается поровну значений (2-я квартиль = медиана)

1. Мода (mode) – наиболее часто встречающееся значение

Размах (range) – разность между максимальным и минимальным значениями

Стандартное отклонение или средне квадратичное отклонение

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Стандарты жизненного цикла Big Data обеспечивают систематический подход к работе с данными, начиная от их сбора и обработки до анализа и принятия решений.

**Вот основные этапы жизненного цикла Big Data:**

1) Сбор данных: В этом этапе происходит сбор больших объемов данных из различных источников, таких как датчики, социальные сети, транзакционные базы данных и т. д. Данные могут быть структурированными, полуструктурированными или неструктурированными.

2) Подготовка данных: После сбора данные требуют предварительной обработки и очистки. Это может включать в себя удаление дубликатов, исправление ошибок, заполнение пропущенных значений и трансформацию данных в подходящий формат для анализа.

3) Хранение данных: На этом этапе данные загружаются в систему хранения данных, такую как хранилище данных или система управления базами данных. Важно выбрать правильную архитектуру хранения данных, учитывая объем и тип данных.

4) Анализ данных: Этот этап включает в себя применение методов Data Mining и аналитических моделей для извлечения полезной информации из данных. Здесь применяются различные алгоритмы и методы, такие как кластеризация, классификация, ассоциативные правила и т. д.

5) Визуализация данных: Результаты анализа данных представляются в удобной форме для визуального анализа. Графики, диаграммы, дашборды и другие визуальные инструменты помогают исследователям и бизнес-пользователям понять и интерпретировать данные.

6) Принятие решений: Инсайты, полученные из анализа данных, используются для принятия решений и определения стратегий. Это может включать принятие бизнес-решений, оптимизацию процессов или прогнозирование будущих событий.

7) Мониторинг и обновление: Жизненный цикл Big Data является итеративным процессом. После принятия решений и внедрения изменений необходимо мониторить результаты и обновлять аналитические модели и методы при необходимости.

**Теперь рассмотрим классификацию методов Data Mining:**

1. Классификация (Classification): Этот метод используется для определения принадлежности объектов к определенным классам на основе заданных характеристик или признаков. Примеры алгоритмов классификации включают решающие деревья, логистическую регрессию и метод опорных векторов.
2. Кластеризация (Clustering): Кластеризация используется для группировки схожих объектов в наборе данных без заранее заданных классов. Алгоритмы кластеризации позволяют обнаружить внутренние структуры данных и выявить скрытые паттерны. Примеры алгоритмов кластеризации включают k-средние, иерархическую кластеризацию и алгоритм DBSCAN.
3. Ассоциативные правила (Association Rules): Этот метод используется для нахождения связей и взаимосвязей между различными элементами в больших наборах данных. Ассоциативные правила находят часто встречающиеся комбинации элементов и помогают идентифицировать скрытые ассоциации. Примером алгоритма для поиска ассоциативных правил является алгоритм Apriori.
4. Регрессия (Regression): Метод регрессии используется для предсказания численных значений на основе зависимостей между переменными. Регрессионные модели позволяют проводить прогнозирование и анализ трендов. Примеры алгоритмов регрессии включают линейную регрессию, регрессию на основе деревьев решений и нейронные сети.
5. Обнаружение выбросов (Outlier Detection): Этот метод используется для выявления аномалий или выбросов в данных, которые сильно отличаются от остальных. Алгоритмы обнаружения выбросов помогают идентифицировать необычные события или аномальные поведения. Примеры алгоритмов обнаружения выбросов включают LOF (Local Outlier Factor) и одноклассовую SVM (Support Vector Machine).

## 31. Для чего нужны гипотезы в анализе данных, какие существуют приемы работы с гипотезами? Понятие уровня статистической достоверности.

Гипотезы в анализе данных представляют собой предположения о связях и закономерностях в данных, которые проверяются на основе статистического анализа. Они необходимы для формулирования предварительных предположений о данных и позволяют исследователям делать выводы о структуре и взаимосвязях в данных.

Для работы с гипотезами используются определенные методы:

· Формулирование гипотез: На этом этапе исследователи выражают свои предположения о взаимосвязях между переменными на основе имеющихся данных

· Сбор данных: Для проверки гипотез собираются соответствующие данные, которые являются основой для анализа.

· Статистический анализ: После сбора данных производится статистический анализ, включающий различные методы, такие как t-тесты, анализ дисперсии, корреляционный анализ и т. д.

· Интерпретация результатов: Полученные результаты статистического анализа анализируются с целью сделать выводы о том, подтверждаются ли или опровергаются предположения, выраженные в гипотезе.

Уровень статистической достоверности оценивает вероятность случайности полученных результатов. Этот уровень обычно измеряется с помощью p-значения, которое показывает вероятность получить такие же или более экстремальные результаты при условии, что нулевая гипотеза верна. Чем меньше p-значение, тем сильнее доказательства против нулевой гипотезы.

## 32. Классификация и кластеризация –понятие и принцип работы. Какие алгоритмы реализуют названные методы анализа данных? Понятия корреляции и регрессии, какие задачи DM можно проводить с их помощью?

**Классификация** относит объекты к заранее определенным категориям или классам на основе их характеристик. Этот метод позволяет автоматизировать процесс группировки данных по категориям и повысить точность анализа. Например, классификация электронных писем на спам и не-спам, или классификация изображений на различные категории (кошки, собаки и т.д.).

**Принцип работы:** Алгоритмы классификации строят модель, которая на основе предоставленных данных предсказывает к какому классу принадлежит новый объект. Эти алгоритмы обучаются на размеченных данных (данных с известными метками классов).

**Примеры алгоритмов:** Логистическая регрессия, метод опорных векторов (SVM), деревья решений, случайный лес, нейронные сети и т.д.

**Кластеризация** объединяет объекты в группы (кластеры) на основе их сходства, таким образом, что объекты внутри одного кластера похожи между собой, а объекты из разных кластеров различны. Каждый кластер объединяет объекты на основе какого-то конкретного критерия, такого как размер, форма, категория или вид. Алгоритмы кластеризации помогают выявить скрытые закономерности, структуру и связи в данных.

**Принцип работы:** Алгоритмы кластеризации не требуют разметки данных и строят кластеры на основе сходства между объектами. Обычно они опираются на меры расстояния или сходства между объектами.

**Примеры алгоритмов**: K-средних, иерархическая кластеризация, самоорганизующиеся карты Кохонена (SOM), DBSCAN, алгоритмы на основе плотности и т.д.

Понятия **корреляции** и **регрессии** тесно связаны с анализом данных и статистикой:

**Корреляция** измеряет степень взаимосвязи или силу связи между двумя переменными. Высокая корреляция указывает на сильную связь между переменными, в то время как низкая корреляция указывает на их слабую связь.

**Задачи DM:** Корреляция помогает определить, какие переменные взаимосвязаны между собой, что может быть полезно для дальнейшего анализа данных или построения моделей.

**Регрессия** используется для прогнозирования значения зависимой переменной на основе одной или нескольких независимых переменных. Она позволяет описывать и анализировать отношения между переменными.

**Задачи DM:** Регрессия используется для прогнозирования или предсказания значений зависимой переменной на основе значений независимых переменных. Например, прогнозирование цены недвижимости на основе ее характеристик, прогнозирование выручки компании на следующий квартал и т.д.

# Методы поисковой оптимизации

## 33. Общая схема эволюционных алгоритмов решения задачи оптимизации функций.

В основе эволюционных алгоритмов лежит принцип «выращивания» лучшего решения на пространстве альтернатив за счет процедур, создающих имитацию биологической эволюции. Наиболее «приспособленный» имеет большую вероятность «выжить». Стандартная схема:

1. Инициализация:

-Начинаем с создания начальной популяции, которая состоит из набора случайных решений (индивидуумов).

-Каждый индивидуум представляет собой потенциальное решение задачи оптимизации и представлен в виде набора параметров.

1. Оценка пригодности:

-Каждый индивидуум оценивается с использованием целевой функции (функции, которую мы пытаемся оптимизировать).

-Это позволяет определить, насколько хорошо каждое решение решает задачу.

1. Отбор:

-Выбираются лучшие индивидуумы (решения) на основе их оценки пригодности.

-Чем лучше решение, тем больше вероятность того, что оно будет выбрано для продолжения процесса.

1. Скрещивание (кроссинговер):

-Выбранные индивидуумы комбинируются для создания новых индивидуумов (потомков).

-Это происходит путем обмена частями решений между выбранными индивидуумами.

1. Мутация:

-В процессе мутации происходит случайное изменение некоторых параметров индивидуумов.

-Это помогает сохранять разнообразие в популяции и предотвращать застревание в локальных оптимумах.

1. Замена:

-Новая популяция заменяет старую популяцию.

-Часто используется стратегия замены по поколениям, когда вся популяция заменяется новой на каждой итерации.

Все вышеперечисленные шаги повторяются до выполнения критерия останова (например, достижение определенного уровня пригодности или достижение максимального числа итераций). Их последовательность может немного меняться в зависимости от алгоритма, например, оценка может производиться только после первой селекции.

После выполнения критерия останова процесс завершается, и лучшее найденное решение возвращается как результат.

## 34. Алгоритм искусственной иммунной системы для решения задачи оптимизации функций.

**Искусственные иммунные системы** (ИИС) - это компьютерные модели, которые используются для решения оптимизационных задач. Они основаны на биологическом принципе иммунной системы, то есть на способности обнаруживать и уничтожать вредные агенты, такие как бактерии и вирусы.

**Аффинность** – термодинамическая характеристика, количественно описывающая силу взаимодействия антигена и антитела. Применительно к искусственным иммунным системам аффинность – это характеристика, оценивающая степень близости (схожести) генетических наборов антигена и антитела. Для искусственных иммунных систем, основанных на бинарном коде, аффинность оценивается с помощью расстояния Хэмминга. Для систем, основанных на вещественном коде – с помощью евклидова расстояния.

**Алгоритм**:

Начало

Шаг 1. Инициализация

Создаём начальную популяцию антител.

Шаг 2. Вычисление аффинности и отбор

Для каждого антитела вычисляем его аффинность к антигену.

Выбираем часть самых эффективных антител, остальные удаляем.

Результаты записываем в массив.

Шаг 3. Клонирование антител

Генерируем копии антител пропорционально их аффинности.

#Чем выше аффинность, тем больше создаётся клонов, и наоборот.

Шаг 4. Модификация антител

Подвергаем копии антител случайным изменениям с вероятностью, обратно пропорциональной их аффинности.

#Чем ниже аффинность, тем выше вероятность мутации.

Возврат к шагу 2

Повторяем процесс до тех пор, пока не достигнут критерий остановки.

Шаг 5. Сохранение в клетках памяти

Сохраняем лучшие антитела